C15

```
/*****************************************************************
 *
 *  This API is an enhancement of the RECOVER_API.  It has been designed to
 *  support restoration of Symmetrix Connect backups as well as network
 *  backups.  In addition, it has been implemented in a client/server
 *  architecture, where the restore GUI on an EDM client can browse, mark, and
 *  initiate restores from the EDM server.  It uses the Restore Engine API
 *  to access, via ONC RPC, the restore functions provided on the EDM server.
 *
 *  This API is defined to allow access to catalog information without
 *  knowing catalog structures or data types.  This API should remain
 *  extendable through the use of data hiding techniques.
 *
 *  Note:
 *  -----
 *  This API uses the philosophy that the caller is responsible for allocation
 *  and freeing of objects.  This should remain consistent throughout.
 *
 *
 *****************************************************************************/
#ifndef H_RESTOREAPI
#define H_RESTOREAPI

#include <eerrno/e_errno.h>
#include <ebcfgapi/BuCfg_Defs.h>          /* Need this for platform types */

#define RESTORE_API_VERSION   1.0

#define TLO_BITFLAG_NETWORK_RESTORE_NOT_POSSIBLE   0x1
/*
 * Datatypes
 */

/* restorable objects are private */
typedef void *restorableObjectPtr;

/* media objects are private */
typedef void *mediaObjectPtr;

/* feedback objects are private */
typedef void *feedbackObjectPtr;

/* EDMProgress object are private */
typedef void *EDMProgressPtr;

/* WIProgress objects are private */
typedef void *WIProgressPtr;

/* Notify objects are private */
typedef void *NotifyObjectPtr;

/* submit objects are private */
typedef void *submitUserObjectPtr;

/* query objects are opaque to the user */
typedef void *queryObjectPtr ;

1 typedef enum {Backup_Good,
```



```
1                Backup_Bad,
1                Backup_Expired,
                 Backup_Child_Without_Data} BackupStatus;

1 typedef enum {Media_Online,
1                Media_Offline,
                 Media_Offsite} MediaStatus;

1 typedef enum {Always_Overwrite,
1                Never_Overwrite,
                 Older_Only_Overwrite} OverwritePolicy;

1 typedef enum {Files_Only,
1                Directories_Only,
1                Others_Only,
                 All_File_Types} FileType;

1 typedef enum {restoreTransportSCSI,
                 restoreTransportNetwork} RestoreTransport;

1 typedef enum {EBREC_AllSizes,
1                EBREC_Equal,
1                EBREC_GreaterThan,
1                EBREC_GreaterOrEqual,
1                EBREC_LessThan,
                 EBREC_LessOrEqual} MatchType;

 typedef char *hostname_ty;

 typedef void *serverHandle;  /* pointer to CSC handle for Restore Engine
                                 server */
 #define MAX_TEMPLATE_LEN 64
 typedef char template_name_ty [MAX_TEMPLATE_LEN];

 typedef struct _EDMRST_submit_args
1{
1  char             *socketClientNm;
1  int              clientSocketPort;
1  char             *mapfile_env;
 }EDMRST_submit_args;




 typedef struct _EBREC_SearchCriteriaRec
1{
1  char       startDirectory[256]; /* Directory to start searching */
1  boolean_ty descendDirectory;    /* Flag to descend into sub dirs */
1  char       searchString[128];
                                   /* String to search for (as in find) */
1  boolean_ty excludeString;
                                   /* Flag to include or exclude string */
1  FileType   typeOfFile;          /* Types of files to search for */
1  char       owner[64];
                                   /* Specific owner of files (or '\0') */
1  boolean_ty excludeOwner;
                                   /* Flag to include or exclude owner */
1  char       group[64];
                                   /* Specific group of files (or '\0') */
1  boolean_ty excludeGroup;
                                   /* Flag to include or exclude group */
1  u_hyper    sizeInBytes;         /* Specific size of files to find */
1  MatchType  sizeMatch;
                                  /* The type of matching to do for size */
```

```
117 1   time_t      startTime;
                                /* First backup date to use for search */
118 1   time_t      endTime;
                                 /* Last backup date to use for search */
119     } EBREC_SearchCriteriaRec;

121     /* Definition of bits in flags input to backup time selection
                                                              functions: */
122     #define BACKUP_SELECTION_FLAG_MASK_COMPLETE       0x1
123     #define BACKUP_SELECTION_FLAG_COMPLETE_ONLY       0x1
124     #define BACKUP_SELECTION_FLAG_PARTIAL_OK          0x0


127     #define INIT_COOKIE    0
128     #define DONE_COOKIE   -1


131     /*************************************************************************
                                                                              *
132      * Definitions to be replaced by Restore Engine Header at some point :
133      *************************************************************************
                                                                              */


135 1   typedef enum {
136 1            RE_STATE_TIMEOUT = -4,
137 1            RE_STATE_ADMIN_QUIT = -3,
138 1            RE_STATE_USER_QUIT = -2,
139 1            RE_STATE_FAILED = -1,
140 1            RE_STATE_SUCCESSFUL = 0,
141 1            RE_STATE_STOPPED,
142 1            RE_STATE_RUNNING,
143 1            RE_STATE_STARTING,
144 1            RE_STATE_PREPHASE,
145 1            RE_STATE_POSTPHASE
146     } RERunningState ;

148     /*********************************************************************
149      * Get Backup Servers:
150      *
151      *  This function is provided to allow retrieval of the
152      *  EDM servers which are available to perform restores from.
153      *
154      *  This is a client-side only function.
155      *
156      * Parameters:
157      *   server (
                 O)  - a pointer to a buffer allocated to receive a host name
158      *   cookie (
                IO) - a place holder for the position in the list of hosts
159      *
160      * NOTE:
161      *       This function will only return a single host name in the 4/99
                                                                    release!
162      *
163      *********************************************************************/

165     eerrno_ty  EDMRST_GetBackupServers(
                                      hostname_ty  server, long *cookie );


168     /*********************************************************************
169      * Initialize:
170      *
171      *  This function initializes a restoral session.  This must be called
172      *  prior to any of the following functions in this API.
173      *
```



```
174      * Parameters:
175      *
176      *  serverName (I) - The machine name of the server to use.
177      *  svrHdl     (
                  O) - A handle to receive a pointer to this user's client
178      *                  handle for the Restore Engine connection.
179      *  timeout    (
                I) - The maximum number of seconds to wait for the connection
180      *                  to the Restore Engine process to be completed.
181      *  rst_mode   (
                 I) - The mode inwhich a restore is going to be done, either
182      *                 with RAW_NETWORK or PLUGIN.
183      *
184      * Return Codes:
185      *      E_SUCCESS                          - operation completed
                                                                 successfully
186      *       EP_RB_RECOVER_BAD_ARGS
187      *       EP_RB_RECOVER_SERVER_FAIL         -If cannot access Dispatch
                                                                 daemon host
188      *                                        or restore engine host within
                                                                 timeout
189      *       EP_RB_RECOVER_NOMEM
190      *       EP_RB_RECOVER_RPC_FAIL            -If server access error
191      *       EP_RB_RECOVER_PARSE_ERROR         -If unable to use config file
192      *       EP_RB_RECOVER_NO_DIRECTORY        -If unable to find EB (
                                                              sub)directory
193      *       EP_RB_RECOVER_FATALERR            - Internal restore eng error
194      *       EP_RB_RECOVER_PERMISSION_DENIED          -If cannot find user
                                                              in password file
195      *       EP_RB_RECOVER_INVALOP             -If another request executing
196      *************************************************************************/
197     eerrno_ty EDMRST_Initialize ( hostname_ty        serverName,
198                                   serverHandle       *svrHdl,
199                                   unsigned long      timeout);

201     /*********************************************************************
202      * Ping:
203      *
204      *  This function allows a ping to be issued in order to keep the
205      *  engine alive and running so that the engine will not time out.
206      *
207      * Parameters:
208      *
209      *  svrHdl (I) - A pointer to this user's client handle for the
210      *               Restore Engine (server) connection.
211      *
212      *********************************************************************/
213     eerrno_ty EDMRST_Ping( serverHandle svrHdl );


216     /*********************************************************************
217      * Finish:
218      *
219      *  This function terminates a restoral session,
                                                 but only during the browse and
220      *  mark phase.
              It will be rejected if a restore is currently being executed.
221      *  This routine will clean up any local memory used in the session
                                                                 and will
222      *  disconnect from the Restore Engine.  After calling this function,
223      *  EDMRST_Initialize MUST be called before calling any other
                                                              functions in this
224      *  API.
225      *
226      * Parameters:
227      *
```

```
228     *  svrHdl (I) - A pointer to this user's client handle for the
229     *                Restore Engine (server) connection.
230     *
231     ***********************************************************************/
232    eerrno_ty EDMRST_Finish( serverHandle svrHdl );


235    /***********************************************************************/
236    /* Storage Allocation/Deallocation Functions                            */
237    /***********************************************************************/


240    /***********************************************************************
241     * Alloc Restorable Objects:
242     *
243     *  This routine allocates space for the given number of restorable
244     *  objects. The caller MUST pass an array of restorable objects.
245     *
246     * Parameters:
247     *  svrHdl             (
                        I) - A pointer to this user's client handle for the
248     *                          Restore Engine (server) connection. (
                                                              must be valid)
249     *  restorableObjects (O) - The array of objects to allocate
250     *  count             (I) - The number of objects to allocate.
251     *
252     * Return Codes:
253     *      E_SUCCESS                - operation completed successfully
254     *       EP_RB_RECOVER_BAD_ARGS
255     *       EP_RB_RECOVER_NOMEM
256     ***********************************************************************/
257    eerrno_ty  EDMRST_AllocRestorableObjects( serverHandle          svrHdl,
258                                              restorableObjectPtr
                                                          *restorableObjs,
259                                            const short          count );

261    /***********************************************************************
262     * Free Restorable Objects:
263     *
264     *  This routine frees the space for the given number of restorable
265     *  objects. The caller MUST pass an array of restorable objects.
266     *
267     * Parameters:
268     *  svrHdl             (
                        I) - A pointer to this user's client handle for the
269     *                          Restore Engine (server) connection. (
                                                              must be valid)
270     *  restorableObjects (IO) - The array of objects to free
271     *  count             (I) - The number of objects to free.
272     *
273     * Return Codes:
274     *      E_SUCCESS                - operation completed successfully
275     *       EP_RB_RECOVER_BAD_ARGS
276     *       EP_RB_RECOVER_INVALOP   - input objType is invalid
277     ***********************************************************************/
278    eerrno_ty  EDMRST_FreeRestorableObjects( serverHandle          svrHdl,
279                                             restorableObjectPtr
                                                          *restorableObjects,
280                                           const short          count);


283    /***********************************************************************
284     * CopyRestorableObject:
285     *
286     *  This routine copies a restorable object allocating dynamic field.
```



```
287     *
288     * Parameters:
289     *  svrHdl             (
                        I) - A pointer to this user's client handle for the
290     *                          Restore Engine (server) connection. (
                                                              must be valid)
291     *  sourceObject      (I) - The restorable object to copy from.
292     *  destinationbject (
                        O) - The pre-allocated restorable object to copy to.
293     *
294     ***********************************************************************/
295    eerrno_ty EDMRST_CopyRestorableObject( serverHandle          svrHdl,
296                                           restorableObjectPtr
                                                          sourceObject,
297                                           restorableObjectPtr
                                                    *destinationObject );


300    /***********************************************************************
301     * Alloc Media Objects:
302     *
303     *  This routine allocates space for the given number of media
304     *  objects. The caller MUST pass an array of media objects.
305     *
306     * Parameters:
307     *  svrHdl        (I) - A pointer to this user's client handle for the
308     *                        Restore Engine (server) connection. (
                                                              must be valid)
309     *  mediaObjects (O) - The array of objects to allocate
310     *  count         (I) - The number of objects to allocate.
311     *
312     * Return Codes:
313     *      E_SUCCESS                - operation completed successfully
314     *       EP_RB_RECOVER_BAD_ARGS
315     *       EP_RB_RECOVER_NOMEM
316     ***********************************************************************/
317    eerrno_ty  EDMRST_AllocMediaObjects( serverHandle   svrHdl,
318                                         mediaObjectPtr *mediaObjects,
319                                        const short    count );

321    /***********************************************************************
322     * Free Media Objects:
323     *
324     *  This routine frees the space for the given number of media
325     *  objects. The caller MUST pass an array of media objects.
326     *
327     * Parameters:
328     *  svrHdl        (
                        I)  - A pointer to this user's client handle for the
329     *                        Restore Engine (server) connection. (
                                                              must be valid)
330     *  mediaObjects (IO) - The array of objects to free
331     *  count         (I)  - The number of objects to free.
332     *
333     * Return Codes:
334     *      E_SUCCESS                - operation completed successfully
335     *       EP_RB_RECOVER_BAD_ARGS
336     *       EP_RB_RECOVER_INVALOP   - input objType is invalid
337     ***********************************************************************/
338    eerrno_ty  EDMRST_FreeMediaObjects( serverHandle   svrHdl,
339                                        mediaObjectPtr *mediaObjects,
340                                       const short    count );

342    /***********************************************************************
343     * Alloc Query Object:
344     *
```

```
 *  This routine allocates space for one query object.
 *
 * Parameters:
 *  svrHdl  (I) - A pointer to this user's client handle for the
 *                Restore Engine (server) connection. (must be valid)
 *  queryObject (O) - Pointer to the query object
 *
 * Return Codes:
 *      E_SUCCESS                - operation completed successfully
 *       EP_RB_RECOVER_BAD_ARGS
 *       EP_RB_RECOVER_NOMEM
 ************************************************************************/
eerrno_ty  EDMRST_AllocQueryObject( serverHandle   svrHdl,
                                    queryObjectPtr *queryObject );

/************************************************************************
 * Free Query Object:
 *
 *  This routine frees the space for one query object.
 *
 * Parameters:
 *  svrHdl      (I)  - A pointer to this user's client handle for the
 *                     Restore Engine (server) connection. (
                                                          must be valid)
 *  queryObject (
          IO) - Ptr to the query object to free -- will be set to NULL
 *
 * Return Codes:
 *      E_SUCCESS                - operation completed successfully
 *       EP_RB_RECOVER_BAD_ARGS
 *       EP_RB_RECOVER_INVALOP   - input objType is invalid
 ************************************************************************/
eerrno_ty  EDMRST_FreeQueryObject( serverHandle   svrHdl,
                                   queryObjectPtr *queryObject );

/************************************************************************
 * Alloc Feedback Object:
 *
 *  This routine allocates space for one feedback object.
 *
 * Parameters:
 *  svrHdl  (I) - A pointer to this user's client handle for the
 *                Restore Engine (server) connection. (Not used)
 *  feedbackObject (O) - The pointer to the allocated object
 *
 * Return Codes:
 *      E_SUCCESS                - operation completed successfully
 *       EP_RB_RECOVER_BAD_ARGS
 *       EP_RB_RECOVER_NOMEM
 ************************************************************************/
eerrno_ty  EDMRST_AllocFeedbackObject( serverHandle   svrHdl,
                                       feedbackObjectPtr
                                                  *feedbackObject );

/************************************************************************
 * Free Feedback Object:
 *
 *  This routine frees the space for one feedback object.
 *
 * Parameters:
 *  svrHdl       (
                 I)  - A pointer to this user's client handle for the
 *                        Restore Engine (server) connection. (
                                                        Not used)
 *  feedbackObjectPtr (
                IO) - The feedback object to free -- wil be set to NULL
```



```
 *
 * Return Codes:
 *      E_SUCCESS                - operation completed successfully
 *       EP_RB_RECOVER_BAD_ARGS
 *       EP_RB_RECOVER_NOMEM
 ************************************************************************/
eerrno_ty  EDMRST_FreeFeedbackObject( serverHandle       svrHdl,
                                      feedbackObjectPtr
                                                  *feedbackObject );

/************************************************************************
 * EDMRST_GetFeedbackStatus
 *
 * Function Description:
 *      Returns the pointer to the string that contains the text
 *      for the provided enum.
 *
 * Parameters:
 *    status (
          I) This is the enum that is returned from a GetFeedbackObject
 *             call to the restore engine
 *
 * Return:
 *      On success: a pointer to a NULL-terminated string which is the
 *                  text for the provided enum;
 *      On failure: a NULL pointer
 ************************************************************************/
char * EDMRST_GetFeedbackStatus(RERunningState status);

/************************************************************************/
/* The following functions obtain configuration information      */

/* from the eb.cfg database                                      */
/************************************************************************/

/************************************************************************
 * Get Source Hosts:
 *
 *  This function is provided to allow retrieval of the
 *  hosts which are restorable by a given user.
 *
 *  Goal:
 *    For a host to be restorable there must have been at least one
 *    successful backup.
 *
 *  The cookie must be initialize to INIT_COOKIE on the first call to
                                                                  this
 *  routine.
       This routine will update the cookie to allow retrieval of more
 *  objects if there is more than "maxEntries".  The cookie will be
 *  returned as DONE_COOKIE when there are no more to retrieve.
 *
 * Parameters:
 *   svrHdl        (
                  I)  - A pointer to this user's client handle for the
 *                          Restore Engine (server) connection.
 *   hostname      (I)  - If NULL, its a no-op. Otherwise, the list of
 *                        recoverable hosts will be filtered based on
 *                        the value of "hostname".
 *   maxEntries    (I)  - the maximum number of hosts to return
 *   hosts         (O)  - a pre-allocated array to return the hosts in
 *   numberEntries (
                   O)  - the real number of hosts returned in the array
 *   cookie        (IO) - a place holder for the list position
 *
 ************************************************************************/
```

```
464 eerrno_ty  EDMRST_GetSourceHosts( serverHandle   svrHdl,
465                                    const char    *hostname,
466                                   const short    maxEntries,
467                                   hostname_ty    *hosts,
468                                   short          *numberEntries,
469                                   long           *cookie );


472 /*********************************************************************
473  * Get Destination Hosts:
474  *
475  *  This function is provided to allow retrieval of the
476  *  hosts which are allowable destinations for the source host
477  *  by a given user.
478  *
479  *  The cookie must be initialize to INIT_COOKIE on the first call to
                                                                    this
480  *  routine.
         This routine will update the cookie to allow retrieval of more
481  *  objects if there is more than "maxEntries".  The cookie will be
482  *  returned as DONE_COOKIE when there are no more to retrieve.
483  *
484  * Parameters:
485  *   svrHdl         (
                  I)  - A pointer to this user's client handle for the
486  *                        Restore Engine (server) connection.
487  *   maxEntries     (I)  - the maximum number of hosts to return
488  *   hosts          (O)  - a pre-allocated array to return the hosts in
489  *   numberEntries (
                   O)  - the real number of hosts returned in the array
490  *   cookie         (IO) - a place holder for the list position
491  *
492  *********************************************************************/
493 eerrno_ty  EDMRST_GetDestinationHosts( serverHandle      svrHdl,
494                                        const short       maxEntries,
495                                        hostname_ty       *hosts,
496                                        short
                                                        *numberEntries,
497                                        long              *cookie );


500 /*********************************************************************
501  * Get Top Level Objects (formerly Get Work Items)
502  *
503  *  This function is provided to allow retrieval of the  work items (
                                                                    for
504  *  network backups) and work item sets (
                                        for Symmetrix Connect backups),
505  *  which are restorable for the given client.
506  *
507  *  It is a GOAL of this routine to return all objects ever backed
508  *  up successfully.  Currently, though, it only looks in the config
509  *  file for 'top level objects' of the given client.
510  *
511  *  The cookie must be initialize to INIT_COOKIE on the first call to
                                                                    this
512  *  routine.
         This routine will update the cookie to allow retrieval of more
513  *  objects if there is more than "maxEntries".  The cookie will be
514  *  returned as DONE_COOKIE when there are no more to retrieve.
515  *
516  * Parameters:
517  *   svrHdl         (
                  I)  - A pointer to this user's client handle for the
518  *                        Restore Engine (server) connection.
519  *   sourceHost     (
```



```
                  I)  - the name of the host whose backups are being restored
520  *   maxEntries     (I)  - the maximum number of objects to return
521  *   topLevObjs     (
                       O)  - a pre-allocated array to return the objects in
522  *   numberEntries (
                     O)  - the real number of objects returned in the array
523  *   cookie         (IO) - a place holder for the list position
524  *
525  *********************************************************************/
526 eerrno_ty  EDMRST_GetTopLevelObjects( serverHandle        svrHdl,
527                                       const char          *sourceHost,
528                                      const short          maxEntries,
529                                      restorableObjectPtr *workItems,
530                                      short
                                                        *numberEntries,
531                                      long                 *cookie );

533 /*********************************************************************
534  * Get All Top Level Objects (formerly Get Work Items)
535  *
536  *  This function is provided to allow retrieval of the  work items (
                                                                    for
537  *  network backups) and work item sets (
                                        for Symmetrix Connect backups),
538  *  which are restorable for the given client.
539  *
540  *  It is a GOAL of this routine to return all objects ever backed
541  *  up successfully.  Currently, though, it only looks in the config
542  *  file for 'top level objects' of the given client.
543  *
544  *  The cookie must be initialize to INIT_COOKIE on the first call to
                                                                    this
545  *  routine.
         This routine will update the cookie to allow retrieval of more
546  *  objects if there is more than "maxEntries".  The cookie will be
547  *  returned as DONE_COOKIE when there are no more to retrieve.
548  *
549  * Parameters:
550  *   svrHdl         (
                  I)  - A pointer to this user's client handle for the
551  *                        Restore Engine (server) connection.
552  *   sourceHost     (
                  I)  - the name of the host whose backups are being restored
553  *   maxEntries     (I)  - the maximum number of objects to return
554  *   topLevObjs     (
                       O)  - a pre-allocated array to return the objects in
555  *   numberEntries (
                     O)  - the real number of objects returned in the array
556  *   cookie         (IO) - a place holder for the list position
557  *
558  *********************************************************************/
559 eerrno_ty  EDMRST_GetAllTopLevelObjects( serverHandle    svrHdl,
560                                          const char       *sourceHost,
561                                         const short       maxEntries,
562                                         restorableObjectPtr *workItems,
563                                         short
                                                        *numberEntries,
564                                         long              *cookie );


567 /*********************************************************************
568  * Get Top Level Object Templates:
569  *
570  *  This function is provided to allow retrieval of the
571  *  templates to which a work item (top level object) belongs.
572  *
```

```
 *  This routine is provided in the event that the goal of the
 *  work-item time routines to include all templates cannot be met.
 *
 *  The cookie must be initialize to INIT_COOKIE on the first call to
                                                                    this
 *  routine.
 *      This routine will update the cookie to allow retrieval of more
 *  objects if there is more than "maxEntries".  The cookie will be
 *  returned as DONE_COOKIE when there are no more to retrieve.
 *
 * Parameters:
 *   svrHdl         (
                  I)  - A pointer to this user's client handle for the
 *                        Restore Engine (server) connection.
 *   topLevelObj    (I)  - the top Level object in question
 *   maxEntries     (I)  - the maximum number of templates to return
 *   templates      (
                 O)  - a pre-allocated array to return the templates in
 *   numberEntries (
                O)  - the real number of templates returned in the array
 *   cookie         (IO) - a place holder for the list position
 *
 ***********************************************************************/
eerrno_ty  EDMRST_GetTopLevelTemplates( serverHandle     svrHdl,
                                        const restorableObjectPtr
                                                          topLevelObj,
                                        const short       maxEntries,
                                        template_name_ty *templates,
                                        short
                                                         *numberEntries,
                                        long              *cookie );

/***********************************************************************
 * Does Alternate Exist:
 *
 *  This routine determines if an alternate trailset exists for the
                                                                   given
 *  template.
 *
 * Parameters:
 *  svrHdl        (I) - A pointer to this user's client handle for the
 *                      Restore Engine (server) connection.
 *  topLevelObj   (I) - the top Level object in question
 *  template      (I) - The name of the template to look for
 *  exists        (
                 O) - Return flag for whether or not the alternate exists
 *  svrHdl        (I) - A pointer to this user's client handle for the
 *                      Restore Engine (server) connection.
 *  topLevelObj   (I) - the top Level object in question
 *  template      (I) - The name of the template to look for
 *  exists        (
                 O) - Return flag for whether or not the alternate exists
 *
 ***********************************************************************/
eerrno_ty EDMRST_DoesAlternateExist( serverHandle            svrHdl,
                                     const restorableObjectPtr
                                                          topLevelObj,
                                     const template_name_ty
                                                          template_name,
                                     boolean_ty             *exists );

/***********************************************************************
 * Set Top Level (formerly Work Item) Template:
 *
 *  This routine sets the template to be used by the specified top
                                                                level
```



```
 *  object (work item, or CBO) and specifies whether or not to use the
 *  alternate trailset.
 *
 * Parameters:
 *  svrHdl     (I) - A pointer to this user's client handle for the
 *                   Restore Engine (server) connection.
 *  workItem   (I) - The top level object to update
 *  template   (I) - The name of the template to use
 *  alternate  (I) - Flag whether or not to use the alternate trailset
 *
 ***********************************************************************/
eerrno_ty EDMRST_SetTopLevelTemplate( serverHandle            svrHdl,
                                      restorableObjectPtr     workItem,
                                     const template_name_ty
                                                          template_name,
                                     const boolean_ty        alternate
                                                                    );



/***********************************************************************
                                                                    **/
/* Backup Configuration Query Functions
                                                                     */
/***********************************************************************
                                                                    **/


/***********************************************************************
 * Get Current Template:
 *
 *  This routine returns the name of the template that is used by
 *       the currently selected top level object (
                                                    work item) and the flag
 *      on whether or not the alternate trail is being used.
 *
 * Parameters:
 *  svrHdl     (I) - A pointer to this user's client handle for the
 *                   Restore Engine (server) connection.
 *  template   (O) - The name of the current template
 *  alternate (
                O) - Flag whether or not the alternate trailset is being used.
 *
 ***********************************************************************/
eerrno_ty EDMRST_GetCurrentTemplate( serverHandle     svrHdl,
                                     template_name_ty template_name,
                                     boolean_ty        *alternate );



/***********************************************************************
 * GetCurrentBackupTime:
 *
 *  This routine returns the time of the backup as selected for the
 *  work item.
 *
 * Parameters:
 *  svrHdl (I) - A pointer to this user's client handle for the
 *                Restore Engine (server) connection.
 *  time   (O) - The time the backup for the work-item was run
 *
 ***********************************************************************/
eerrno_ty EDMRST_GetCurrentBackupTime( serverHandle svrHdl,
                                       time_t       *time );

/***********************************************************************/
/* Backup Time Selection Functions
                                                                     */
/***********************************************************************/
```

```
685  /*****************************************************************
686   * SetPrevBackup
687   *
688   *  This routine sets the recover environment to the previous backup
689   *  of the currently selected work item
690   *
691   *  Goal (?):
692   *    This includes both primary and alternate trailsets for all
693   *    templates for which this work-item has backups.
694   *
695   * Parameters:
696   *  svrHdl (I) - A pointer to this user's client handle for the
697   *                Restore Engine (server) connection.
698   *  flags  (
           I) - Selection Flags: e.g., Complete backups only/partial ok
699   *
700   *****************************************************************/
701  eerrno_ty  EDMRST_SetPrevBackup( serverHandle    svrHdl,
702                                   u_long                flags );

704  /*****************************************************************
705   * SetNextBackup
706   *
707   *  This routine sets the recover environment to the the next backup
708   *  of the specified work item.
709   *
710   *  Goal:
711   *    This includes both primary and alternate trailsets for all
712   *    templates for which this work-item has backups.
713   *
714   * Parameters:
715   *  svrHdl (I) - A pointer to this user's client handle for the
716   *                Restore Engine (server) connection.
717   *  flags  (
           I) - Selection Flags: e.g., Complete backups only/partial ok
718   *
719   *****************************************************************/
720  eerrno_ty  EDMRST_SetNextBackup( serverHandle    svrHdl,
721                                   u_long                flags );

723  /*****************************************************************
724   * SetFirstBackup
725   *
726   *  This routine sets the recover environment to the first backup
727   *  kept for the specified work item.
728   *
729   *  Goal:
730   *    This includes both primary and alternate trailsets for all
731   *    templates for which this work-item has backups.
732   *
733   * Parameters:
734   *  svrHdl (I) - A pointer to this user's client handle for the
735   *                Restore Engine (server) connection.
736   *  flags  (
           I) - Selection Flags: e.g., Complete backups only/partial ok
737   *
738   *****************************************************************/
739  eerrno_ty  EDMRST_SetFirstBackup( serverHandle   svrHdl,
740                                    u_long        flags );

742  /*****************************************************************
743   * SetMostRecentBackup
744   *
745   *  This routine sets the recover environment to the most recent
                                                                backup
```



```
746   *  of the specified work item.
747   *
748   *  Goal:
749   *    This includes both primary and alternate trailsets for all
750   *    templates for which this work-item has backups.
751   *
752   * Parameters:
753   *  svrHdl (I) - A pointer to this user's client handle for the
754   *                Restore Engine (server) connection.
755   *  flags  (
           I) - Selection Flags: e.g., Complete backups only/partial ok
756   *
757   *****************************************************************/
758  eerrno_ty  EDMRST_SetMostRecentBackup(   serverHandle    svrHdl,
759                                           u_long           flags );

761  /*****************************************************************
762   * Get All Backup Times:
763   *
764   *  This function is provided to allow retrieval of the times which
765   *  the given work-item was backed up.
766   *
767   *  The cookie must be initialize to INIT_COOKIE on the first call to
                                                                    this
768   *  routine.
           This routine will update the cookie to allow retrieval of more
769   *  objects if there is more than "maxEntries".  The cookie will be
770   *  returned as DONE_COOKIE when there are no more to retrieve.
771   *
772   * Parameters:
773   *   svrHdl         (
                    I)  - A pointer to this user's client handle for the
774   *                          Restore Engine (server) connection.
775   *   startTime      (I)  - first time which should be returned (
                                                          nothing before)
776   *   endTime        (I)  - last time which should be returned (
                                                           nothing after)
777   *   flags          (
                  I)  - Backup constraint flags: e.g. full-only/partial-ok
778   *   maxEntries     (I)  - the maximum number of templates to return
779   *   times          (O)  - a pre-allocated array to return the times in
780   *   numberEntries (
                 O)  - the real number of templates returned in the array
781   *   cookie         (IO) - a place holder for the list position
782   *
783   *****************************************************************/
784  eerrno_ty  EDMRST_GetAllBackupTimes( serverHandle svrHdl,
785                                       const time_t startTime,
786                                       const time_t endTime,
787                                        u_long       flags,
788                                       const short maxEntries,
789                                       time_t *times,
790                                       short *numberEntries,
791                                       long *cookie );

793  /*****************************************************************
794   * Set Backup for Time
795   *
796   *  This routine sets the recover environment to the given backup time
797   *
798   *  Goal:
799   *    This includes both primary and alternate trailsets for all
800   *    templates for which this work-item has backups.
801   *
802   * Parameters:
803   *  svrHdl (I) - A pointer to this user's client handle for the
```

```
804  *                  Restore Engine (server) connection.
805  *  time   (I) - The time desired
806  *  flags  (
                I) - Selection Flags: e.g., Complete backups only/partial ok
807  *
808  ************************************************************************/
809 eerrno_ty  EDMRST_SetBackupForTime( serverHandle svrHdl,
810                                     const time_t time,
811                                     u_long       flags );


814 /************************************************************************
815  * Get Restorable Objects:
816  *
817  *  These functions are provided to allow retrieval of the
818  *  child objects which are restorable given the parent object. The
819  *  caller specifies the parent object and whether or not
820  *  to include bad files.
821  *
822  *  The cookie must be initialize to INIT_COOKIE on the first call to
                                                                    this
823  *  routine.
           This routine will update the cookie to allow retrieval of more
824  *  objects if there is more than "maxEntries".  The cookie will be
825  *  returned as DONE_COOKIE when there are no more to retrieve.
826  *
827  * Parameters:
828  *   svrHdl           (
                   I)  - A pointer to this user's client handle for the
829  *                        Restore Engine (server) connection.
830  *   parentObject   (I)  - the parent object
831  *   allowBadFiles  (I)  - flag whether or not to include bad files
832  *   maxEntries     (I)  - the maximum number of objects to return
833  *   objects        (
                     O)  - a pre-allocated array to return the objects in
834  *   numberEntries  (
                   O)  - the real number of objects returned in the array
835  *   cookie         (IO) - a place holder for the list position
836  *
837  ************************************************************************/
838 eerrno_ty  EDMRST_GetRestorableObjects( serverHandle        svrHdl,
839                                         const restorableObjectPtr
                                                        parentObject,
840                                        const boolean_ty
                                                        allowBadFiles,
841                                        const long          maxEntries,
842                                        restorableObjectPtr *objects,
843                                        long
                                                        *numberEntries,
844                                        long                *cookie );


847 /************************************************************************
848  * Restorable Object Access Routines:
849  *
850  *  These routines retrieve information pertinent to a given
                                                        restorable object.
851  *
852  * Notes:
853  *
854  *  GetObjectSize       - returns the size of the individual object
855  *
856  * Parameters:
857  *
858  *  svrHdl      (I) - A pointer to this user's client handle for the
859  *                    Restore Engine (server) connection.
```



```
860  *  thisObject (I) - The restoral object
861  *
862  ************************************************************************/

864 const char *EDMRST_GetObjectBaseName( serverHandle        svrHdl,
865                                        restorableObjectPtr thisObject
                                                                    );

867 const char *EDMRST_GetObjectFullName( serverHandle        svrHdl,
868                                        restorableObjectPtr thisObject
                                                                    );

870 boolean_ty EDMRST_IsObjectTopLevel( serverHandle        svrHdl,
871                                     restorableObjectPtr thisObject );

873 boolean_ty EDMRST_IsObjectContainer( serverHandle        svrHdl,
874                                      restorableObjectPtr thisObject );

876 boolean_ty EDMRST_IsObjectLeaf( serverHandle        svrHdl,
877                                 restorableObjectPtr thisObject );

879 const char *EDMRST_GetObjectTypeString( serverHandle        svrHdl,
880                                         restorableObjectPtr
                                                        thisObject );

882 u_long EDMRST_GetObjectPermissions( serverHandle        svrHdl,
883                                     restorableObjectPtr thisObject );

885 const char *EDMRST_GetObjectOwnerString( serverHandle        svrHdl,
886                                          restorableObjectPtr
                                                        thisObject );

888 const char *EDMRST_GetObjectGroupString( serverHandle        svrHdl,
889                                          restorableObjectPtr
                                                        thisObject );

891 time_t EDMRST_GetObjectModDate( serverHandle        svrHdl,
892                                 restorableObjectPtr thisObject );

894 u_hyper EDMRST_GetObjectSize( serverHandle        svrHdl,
895                               restorableObjectPtr thisObject );

897 BackupStatus  EDMRST_GetObjectStatus( serverHandle        svrHdl,
898                                       restorableObjectPtr thisObject
                                                                    );

900 char EDMRST_GetWorkItemType( serverHandle        svrHdl,
901                               restorableObjectPtr thisObject );

903 const char *EDMRST_GetWorkItemBIC( serverHandle        svrHdl,
904                                     restorableObjectPtr thisObject );

906 /************************************************************************
907  * Is Object Markable
908  *
909  *  This routine determines if the current user is able to restore the
910  *  given object.
911  *
912  * Parameters:
913  *
914  *  svrHdl      (I) - A pointer to this user's client handle for the
915  *                    Restore Engine (server) connection.
916  *  thisObject (I) - The restoral object
917  *  markable   (O) - TRUE/FALSE indicating markable or not
918  *
919  ************************************************************************/
```

```
920  eerrno_ty EDMRST_IsObjectMarkable( serverHandle          svrHdl,
921                                     const restorableObjectPtr
                                                              thisObject,
922                                     boolean_ty           *markable );

924  /*****************************************************************
925   * Is Object Searchable
926   *
927   *  This routine determines if a top level object supports the Find
                                                                  function.
928   *
929   * Parameters:
930   *
931   *  svrHdl     (I) - A pointer to this user's client handle for the
932   *                    Restore Engine (server) connection.
933   *  thisObject (I) - The restoral object to query
934   *  searchable (O) - TRUE/FALSE indicating searchable or not
935   *
936   * Return Codes:
937   *      E_SUCCESS                - operation completed successfully
938   *       EP_RB_RECOVER_BAD_ARGS
939   *       EP_RB_RECOVER_RPC_FAIL
940   *       EP_RB_RECOVER_SERVER_FAIL
941   *       EP_RB_RECOVER_INVALOP           -If another request active
942   *
943   *****************************************************************
                                                                     */

945  eerrno_ty EDMRST_IsObjectSearchable( serverHandle        svrHdl,
946                                       const restorableObjectPtr
                                                              thisObject,
947                                       boolean_ty
                                                            *searchable );

949  /*****************************************************************/


952  /*****************************************************************
953   * Get Symmetrix Support
954   *
955   *  This routine determines if a top level object supports restoral
                                                            thru a Symm
956   *
957   * Parameters:
958   *
959   *  svrHdl     (I) - A pointer to this user's client handle for the
960   *                    Restore Engine (server) connection.
961   *  thisObject (I) - The restoral object to query
962   *  symm_restorable (O) - TRUE/FALSE indicating restorable over Symm (
                                                                  SCSI)
963   *
964   * Return Codes:
965   *      E_SUCCESS                - operation completed successfully
966   *       EP_RB_RECOVER_BAD_ARGS
967   *       EP_RB_RECOVER_RPC_FAIL
968   *       EP_RB_RECOVER_SERVER_FAIL
969   *       EP_RB_RECOVER_INVALOP           -If another request active
970   *
971   *****************************************************************
                                                                     */

973  eerrno_ty EDMRST_GetSymmRestoreOption( serverHandle      svrHdl,
974                                         const restorableObjectPtr
                                                              thisObject,
975                                         boolean_ty
                                                          *symm_restorable );
```



```
977  /*****************************************************************
978   * Get Network Support
979   *
980   *  This routine determines if a top level object supports restoral
                                                                  thru
981   *  the network
982   *
983   * Parameters:
984   *
985   *  svrHdl     (I) - A pointer to this user's client handle for the
986   *                    Restore Engine (server) connection.
987   *  thisObject (I) - The restoral object to query
988   *  net_restorable (O) - TRUE/FALSE indicating restorable over Symm (
                                                                  SCSI)
989   *
990   * Return Codes:
991   *      E_SUCCESS                - operation completed successfully
992   *       EP_RB_RECOVER_BAD_ARGS
993   *       EP_RB_RECOVER_RPC_FAIL
994   *       EP_RB_RECOVER_SERVER_FAIL
995   *       EP_RB_RECOVER_INVALOP           -If another request active
996   *
997   *****************************************************************
                                                                     */


999  eerrno_ty EDMRST_GetNetworkRestoreOption( serverHandle   svrHdl,
1000                                         const restorableObjectPtr
                                                              thisObject,
1001                                        boolean_ty
                                                          *net_restorable );

1003 /*****************************************************************/

1005 /*****************************************************************
1006  * Find Routines:
1007  *
1008  *  These routines allow the user to find restorable objects.
                                                              Returned
1009  *  is an array of found objects and an array of backup times
                                                            associated
1010  *  with the objects.  These arrays are 1-to-1.  That is,
                                                          the nth Object
1011  *  was backed up at the nth time.
1012  *
1013  *  This operation is performed asynchronously by the Restore Engine,
                                                               and the
1014  *  first API function, EDMRST_FindRestorableObjects,
                                                        is used to start the
1015  *  'find'.
      EDMRST_GetFindResults is used to test for completion of the find,
1016  *  and receive the results (parts of, at least) if it is done.
1017  *
1018  * EDMRST_FindRestorableObjects Parameters:
1019  *
1020  *   svrHdl          (
                        I) - A pointer to this user's client handle for the
1021  *                        Restore Engine (server) connection.
1022  *   searchCriteria (I) - The criteria used for the search
1023  *
1024  *****************************************************************/
1025 eerrno_ty EDMRST_FindRestorableObjects(
                                        serverHandle            svrHdl,
1026                                      EBREC_SearchCriteriaRec
                                                      *searchCriteria);
```

```
1028 /*****************************************************************
1029  * EDMRST_GetFindResults Parameters:
1030  *
1031  *   svrHdl          (
                        I)  - a pointer to this user's client handle for the
1032  *                           Restore Engine (server) connection.
1033  *   interrupt      (I)  - requests cancellation of the find (if TRUE)
1034  *   maxEntries     (
                        I)  - the maximum number of found objects to return
1035  *   objects        (
                       O)  - a pre-allocated array to return the objects in
1036  *   times           (
                  O)  - a pre-allocated array to return the backup times in
1037  *   numberEntries (
                     O)  - the real number of objects returned in the array
1038  *   cookie         (IO) - a place holder for the list position
1039  *
1040  *****************************************************************/
1041 eerrno_ty EDMRST_GetFindResults( serverHandle          svrHdl,
1042                                  boolean_ty           interrupt,
1043                                 const long           maxEntries,
1044                                 restorableObjectPtr *foundObjects,
1045                                 time_t              *times,
1046                                 long                *numberEntries,
1047                                 long                *cookie );

1049 /*****************************************************************
1050  * MarkObject() and GetMarkResults()
1051  *
1052  *   MarkObject is passed a restorableObject and marks files for
1053  *   recovery based on the input criteria.
                                    It starts an asynchronous operation
1054  *   in the Restore Engine to perform the marking, and returns.
1055  *
1056  *   GetMarkResults is called to test for completion of the mark
                                                          operation,
1057  *   and receive results when it is done.
                                       It can also be used to interrupt
1058  *   the mark operation.
1059  *
1060  * MarkObject Parameters:
1061  *
1062  *  svrHdl          (
                       I) - A pointer to this user's client handle for the
1063  *                           Restore Engine (server) connection.
1064  *  thisObject      (I) - The restoral object;
                                          can be a leaf object (e.g. a
1065  *                           file), or a container object (
                                                 e.g., a directory).
1066  *  time           (I) - (
                        optional) the backup time to perform the mark on --
1067  *                           if not specified,
                                       uses currrently selected backup; if
1068  *                           specified,
                                    leaves selected backup time unchanged
1069  *  allowBadfiles (I) - allows marking of files of state BADDATA.
1070  *  descend         (
                 I) - Should mark operation descend to operate on the content
1071  *                           of container objects.
1072  *
1073  *****************************************************************/

1075 eerrno_ty EDMRST_MarkObject( serverHandle         svrHdl,
1076                              restorableObjectPtr thisObject,
1077                              time_t              time,
1078                               boolean_ty          allowBadFiles,
```



```
1079                                  boolean_ty          descend );

1081 /*****************************************************************
1082  *
1083  * GetMarkResults Parameters:
1084  *
1085  *  svrHdl          (
                        I) - A pointer to this user's client handle for the
1086  *                           Restore Engine (server) connection.
1087  *  interrupt      (I) - requests cancellation of the mark (if TRUE)
1088  *                           WARNING: If the operation is aborted,
                                                      the mark will be
1089  *                           left in an unknown state.  That is,
                                                        any one of the
1090  *                           descendants of the marked object may be
                                                          marked or not.
1091  *                           It is up to the caller to determine how to
                                                               proceed
1092  *                           afterwards.
1093  *  BadFilesCount (O) - returns the file count with BADDATA.
1094  *  PermDenyFilesCount (
                          O) -- returns the file count with permission denied.
1095  *  fileMarked     (
                 O) - return the total files marked after this mark occurred.
1096  *  dirMarked      (
                    O) - return the total directories marked after this mark
1097  *                           occurred.
1098  *  otherMarked    (
                   O) - return the total "other" files marked after this mark.
1099  *
1100  *****************************************************************
                                                                    **/

1102 eerrno_ty EDMRST_GetMarkResults( serverHandle         svrHdl,
1103                                  boolean_ty          interrupt,
1104                                  u_long              *BadFilesCount,
1105                                  u_long
                                                    *PermDenyFilesCount,
1106                                  u_long              *fileMarked,
1107                                  u_long              *dirMarked,
1108                                  u_long              *otherMarked );

1110 /*****************************************************************
1111  * UnmarkObject() and GetUnmarkResults()
1112  *
1113  *   UnmarkObject operates like MarkObject,
                                          in that it is supported through
1114  *   two API calls -- UnmarkObject and GetUnmarkresults.
                                          Unmark starts an
1115  *   asynchronous operation in the Restore Engine to perform the
                                                        unmarking,
1116  *   and returns.
1117  *
1118  *   GetUnmarkResults is called to test for completion of the unmark
                                                        operation,
1119  *   and receive results when it is done.
                                       It can also be used to interrupt
1120  *   the unmark operation.
1121  *
1122  * UnmarkObject Parameters:
1123  *
1124  *  svrHdl          (
                        I) - A pointer to this user's client handle for the
1125  *                           Restore Engine (server) connection.
1126  *  thisObject      (I) - The restoral object;
                                          can be a leaf object (e.g. a
```

```
 *                          file), or a container object (
                                                        e.g., a directory).
 *  time          (I) - (
                    optional) the backup time to perform the unmark on --
 *                          if not specified,
                                    uses currrently selected backup; if
 *                          specified,
                                    leaves selected backup time unchanged
 *  BadFilesOnly  (
                   I) - allows unmarking ONLY of files of state BADDATA.
 *  descend         (
                  I) - Should unmark operation descend to operate on the
 *                          content of container objects.
 ************************************************************************
                                                                      **/

eerrno_ty EDMRST_UnmarkObject( serverHandle        svrHdl,
                               restorableObjectPtr thisObject,
                              time_t               time,
                               boolean_ty          BadFilesOnly,
                               boolean_ty          descend );

/*********************************************************************
 *
 * GetUnmarkResults Parameters:
 *
 *  svrHdl          (
                   I) - A pointer to this user's client handle for the
 *                          Restore Engine (server) connection.
 *  interrupt      (I) - requests cancellation of the unmark (if TRUE)
 *                          WARNING: If the operation is aborted,
                                                        the unmark will
 *                          be left in an unknown state.  That is,
                                                         any one of the
 *                          descendants of the unmarked object may be
                                                            marked or
 *                          not.
                               It is up to the caller to determine how to
 *                          proceed afterwards.
 *  BadFilesCount (O) - returns the file count with BADDATA.
 *  fileMarked     (
           O) - return the total files marked after this mark occurred.
 *  dirMarked      (
             O) - return the total directories marked after this mark
 *                        occurred.
 *  otherMarked    (
            O) - return the total "other" files marked after this mark.
 *
 *********************************************************************/

eerrno_ty EDMRST_GetUnmarkResults( serverHandle svrHdl,
                                    boolean_ty   interrupt,
                                   u_long       *BadFilesCount,
                                   u_long       *fileMarked,
                                   u_long       *dirMarked,
                                   u_long       *otherMarked );

/*********************************************************************
 * IsObjectMarked:
 *
 *  This function determines if each of a set of restorable objects
                                                            has been
 *  marked for restoration.
                        It is intended to allow the user to determine the
 *  current restore markings for the restorable objects at the same
                                                          object tree
```



```
 *  hierarchy level,
                  i.e. objects that have the same parent restorableObject.
 *
 * Parameters:
 *
 *  svrHdl        (I) - A pointer to this user's client handle for the
 *                      Restore Engine (server) connection.
 * numEntries     (I) - number of objects in bufPtrArray to be checked
 * bufPtrArray    (I) - restorable objects to be checked for marks
 * numChecked     (O) - number of objects in bufPtrArray found marked
 * marked         (O) - array of booleans to receive individual marked(
                                                                    1) /
 *                      unmarked(
                          0) result for corresponding bufPtrArray entry
 *********************************************************************/

eerrno_ty EDMRST_IsObjectMarked( serverHandle         svrHdl,
                                  const unsigned long numEntries,
                                 restorableObjectPtr *bufPtrArray,
                                  unsigned long       *numChecked,
                                  boolean_ty          *marked );

/*********************************************************************/

eerrno_ty EDMRST_GetMarkedTotalSize( serverHandle svrHdl,
                                     u_hyper *totalSize );

/*********************************************************************
 * Get Necessary Media:
 *
 *  This function is provided to allow retrieval of the
 *  media necessary to restore the currently marked objects.
 *
 *  The cookie must be initialize to INIT_COOKIE on the first call to
                                                                  this
 *  routine.
      This routine will update the cookie to allow retrieval of more
 *  objects if there is more than "maxEntries".  The cookie will be
 *  returned as DONE_COOKIE when there are no more to retrieve.
 *
 * Parameters:
 *  svrHdl          (
                  I) - a pointer to this user's client handle for the
 *                       Restore Engine (server) connection.
 *  maxEntries    (I) - the maximum number of media objects to return
 *  objects       (O) - a pre-allocated array to return the objects in
 *  numberEntries (
           O) - the real number of media objects returned in the array
 *  all           (i) - get all media, or necassary
 *  cookie       (IO) - a place holder for the list position
 *
 *********************************************************************/
eerrno_ty EDMRST_GetNecessaryMedia( serverHandle   svrHdl,
                                     const short    maxEntries,
                                    mediaObjectPtr *objects,
                                    short          *numberEntries,
                                     boolean_ty     all,
                                    long           *cookie );

/*********************************************************************
 * Media Object Access Routines:
 *
 *  These routines retrieve information pertinent to a given Media
                                                                object.
 *
 * Parameters:
```

```
 *
 *  svrHdl      (I) - (
                ignored) A pointer to this user's client handle for the
 *                    Restore Engine (server) connection.
 *  thisObject (I) - The media object
 *
 ****************************************************************/
const char *EDMRST_GetMediaVolid( serverHandle   svrHdl,
                                  mediaObjectPtr thisObject );

long EDMRST_GetMediaSequenceNumber( serverHandle   svrHdl,
                                    mediaObjectPtr thisObject );

const char *EDMRST_GetMediaBarcodeString( serverHandle   svrHdl,
                                    mediaObjectPtr  thisObject );

const char *EDMRST_GetMediaTypeDescrip( serverHandle   svrHdl,
                                  mediaObjectPtr thisObject );

const char *EDMRST_GetMediaTrail( serverHandle   svrHdl,
                                  mediaObjectPtr thisObject );

const char *EDMRST_GetMediaTypeToken( serverHandle   svrHdl,
                                  mediaObjectPtr thisObject );

const char *EDMRST_GetMediaComments( serverHandle   svrHdl,
                                  mediaObjectPtr thisObject );

const char *EDMRST_GetMediaLocation( serverHandle   svrHdl,
                                  mediaObjectPtr thisObject );

MediaStatus  EDMRST_GetMediaStatus( serverHandle   svrHdl,
                                    mediaObjectPtr thisObject );

uchar_t EDMRST_GetMediaSide( serverHandle   svrHdl,
                             mediaObjectPtr thisObject );



/*****************************************************
 * Duplicate Media Access Routines
 * Inputs: Svr Handle - see above
 *         dup_number: the number of the duplicate wanted
 *         thiObject:  The media object to get the dups for...
 *****************************************************
 */
short EDMRST_GetNumberOfDuplicates( serverHandle   svrHdl,
                                    mediaObjectPtr thisObject );


const char *
EDMRST_GetDuplicateVolid( serverHandle   svrHdl,
                          int dup_number,
                          mediaObjectPtr thisObject );

long
EDMRST_GetDuplicateSequenceNumber( serverHandle   svrHdl,
                                   int dup_number,
                                   mediaObjectPtr thisObject );

const char *
EDMRST_GetDuplicateBarcodeString( serverHandle   svrHdl,
                                  int dup_number,
                                  mediaObjectPtr thisObject );

const char *
```



```
EDMRST_GetDuplicateTypeDescrip( serverHandle   svrHdl,
                                int dup_number,
                                mediaObjectPtr thisObject );


const char *
EDMRST_GetDuplicateTypeToken( serverHandle   svrHdl,
                              int dup_number,
                              mediaObjectPtr thisObject );


MediaStatus
EDMRST_GetDuplicateStatus( serverHandle   svrHdl,
                           int dup_number,
                           mediaObjectPtr thisObject );
const char *
EDMRST_GetDuplicateLocation( serverHandle   svrHdl,
                             int dup_number,
                             mediaObjectPtr thisObject );

/*****************************************************************
 * Restoral Management Functions:
 *
 *  These functions are provided to allow:
 *       - creation of submit objects,
 *         which define the set of objects to be
 *         restored and the scripts to be run before and after
 *         restoration,
 *       - starting the restoral of a submit object.
 *       - polling of the status of an ongoing restore,
 *         including the ability to
 *         interrupt the restore,
 *         and to receive information necessary to
 *         query the user for input needed for the pre-restore or
 *         post-restore
 *         scripts, suspending, restarting,
 *       - supply of user responses to pre- and post- restore script
 *         queries
 *
 * The following functions comprise restoral management:
 *
 *      EDMRST_Submit
 *      EDMRST_GetSubmitResults
 *       EDMRST_Start
 *       EDMRST_GetRestoreFeedback
 *       EDMRST_GetQuestion
 *       EDMRST_SetUserAnswer
 *
 *****************************************************************
 * Submit
 *
 * This function starts the creation or update of a submit object from
 *         the
 * currently marked restorable objects.
 *                 Its completion is tested for with
 * EDMRST_GetSubmitResults.
 *                 The returned submit object ID is passed to
 * EDMRST_Start to begin execution of the restore.
 *
 * Parameters:
 *
 * svrHdl       (I) - A pointer to this user's client handle for
 *                    the Restore Engine (server) connection.
 * policy       (I) - The overwrite policy to use
 * inPlace      (
               I) - Flag if the restoral is to be in original locations
```

```
1354    * hostName      (I) - host to restore to (only if inPlace == False)
1355    * directory     (I) - directory to restore to (
                                                only if inPlace == False)
1356    * transport     (
              I) - Indicator of transport the restoral is to be over (SCSI
1357    *                    or network)
1358    * submitObjID   (
              I) - ID of an existing submit object which is to be added to
1359    *
1360    * Return Codes:
1361    *       E_SUCCESS                - operation completed successfully
1362    *        EP_RB_RECOVER_BAD_ARGS
1363    *        EP_RB_RECOVER_NOMEM
1364    *        EP_RB_RECOVER_RPC_FAIL
1365    *        EP_RB_RECOVER_SERVER_FAIL
1366    *        EP_RB_RECOVER_INVALOP          -If another request still
                                                              executing
1367    ********************************************************************/
1368   eerrno_ty EDMRST_Submit( serverHandle           svrHdl,
1369                            const char             *hostName,
1370                            const OverwritePolicy  policy,
1371                            const boolean_ty       inPlace,
1372                            const char             *directory,
1373                           const RestoreTransport transport,
1374                           unsigned int           submitObjID,
1375                            EDMRST_submit_args    *submitArgs);

1377   /*******************************************************************
1378    * GetSubmitResults
1379    *
1380    * This function tests for completion of an EDMRST_Submit call,
                                                              with the
1381    * option of cancelling the submit.
1382    *
1383    * Parameters:
1384    *
1385    * svrHdl         (I) - A pointer to this user's client handle for
1386    *                      the Restore Engine (server) connection.
1387    * interrupt      (I) - Flag if the submit is to be canceled
1388    * submitObjID    (
                  O) - ID of the submit object which describes the restore
1389    * objectsDone    (
              O) - number of objects -- total number in the submit object
1390    *                      if operation is complete,
                                    or number processed so far if
1391    *                      submit operation is still executing (
                                                  INCOMPLETE status)
1392    *
1393    * Return Codes:
1394    *       E_SUCCESS                - operation completed successfully
1395    *        EP_RB_RECOVER_RPC_INCOMPLETE    -If submit not done yet
1396    *        EP_RB_RECOVER_ABORT             -If submit successfully
                                                        interrupted
1397    *        EP_RB_RECOVER_FATALERR          -If submit failed.
                                                  Internal restore error
1398    *        EP_RB_RECOVER_BAD_ARGS
1399    *        EP_RB_RECOVER_NOMEM
1400    *        EP_RB_RECOVER_RPC_FAIL
1401    *        EP_RB_RECOVER_SERVER_FAIL
1402    *        EP_RB_RECOVER_INVALOP          -If another request still
                                                              executing
1403    ********************************************************************/
1404   eerrno_ty EDMRST_GetSubmitResults( serverHandle     svrHdl,
1405                                      const boolean_ty interrupt,
1406                                      unsigned int     *submitObjID,
1407                                      unsigned long    *objectsDone );
```



```
1410   /*******************************************************************
1411    * Start
1412    *
1413    * This function begins execution of the restoral of the objects in a
1414    * submit object.  Its progress and requests for operator input are
1415    * received via EDMRST_GetRestoreFeedback.
1416    *
1417    * Parameters:
1418    *
1419    * svrHdl         (I) - A pointer to this user's client handle for
1420    *                      the Restore Engine (server) connection.
1421    * submitObjID    (
                    I) - ID of the submit object that describes the restore
1422    *
1423    * Return Codes:
1424    *       E_SUCCESS                - operation completed successfully
1425    *        EP_RB_RECOVER_BAD_ARGS
1426    *        EP_RB_RECOVER_NOMEM
1427    *        EP_RB_RECOVER_RPC_FAIL
1428    *        EP_RB_RECOVER_SERVER_FAIL
1429    *        EP_RB_RECOVER_INVALOP          -If another request still
                                                              executing
1430    ********************************************************************/
1431   eerrno_ty EDMRST_Start( serverHandle  svrHdl,
1432                           unsigned int submitObjID );

1434   /*******************************************************************
1435    * GetRestoreFeedback
1436    *
1437    * This function is used to poll for the status of an ongoing restore,
                                                                  and
1438    * includes the ability to interrupt the restore,
                                                  and to receive information
1439    * necessary to query the user for input needed for the pre-restore or
1440    * post-restore scripts.
1441    *
1442    * Parameters:
1443    *
1444    * svrHdl         (I) - A pointer to this user's client handle for
1445    *                      the Restore Engine (server) connection.
1446    * quitRestore    (I) - Flag if the restoral is to be stopped
1447    * currentState   (
              O) - Pointer to storage to receive the state of the restore
1448    * feedbackPtr    (
              IO) - Pointer to structure to receive restore feedback data
1449    *
1450    * Return Codes:
1451    *       E_SUCCESS                - If restore operation completed
                                                              successfully
1452    *        EP_RB_RECOVER_RPC_INCOMPLETE    -If restore not done yet
1453    *        EP_RB_RECOVER_ABORT             -If restore successfully
                                                        interrupted
1454    *        EP_RB_RECOVER_EXECUTEFAILED     -If restore execution failed
1455    *        EP_RB_RECOVER_PREFAILED                -If pre-restore
                                                  script execution failed
1456    *        EP_RB_RECOVER_POSTFAILED        -If post-restore script
                                                        execution failed
1457    *        EP_RB_RECOVER_CLIENT_FAIL       -If problem with restore
                                                        target host
1458    *        EP_RB_RECOVER_BAD_ARGS
1459    *        EP_RB_RECOVER_NOMEM
1460    *        EP_RB_RECOVER_RPC_FAIL
1461    *        EP_RB_RECOVER_SERVER_FAIL
1462    *        EP_RB_RECOVER_INVALOP          -If another request still
```

```
                                                                    executing
 ********************************************************************/

eerrno_ty EDMRST_GetRestoreFeedback( serverHandle     svrHdl,
                                     const boolean_ty quitRestore,
                                     RERunningState   *currentState,
                                     feedbackObjectPtr feedbackPtr );

/*********************************************************************
 * GetQuestion
 *
 * This function is used to fetch the data needed to query the user
                                                                  during a
 * pre-restore or post-restore script execution.
 *
 * Parameters:
 *
 * svrHdl   (I) - A pointer to this user's client handle for
 *                the Restore Engine (server) connection.
 * queryPtr (O) - Pointer to the object containing the question data.
 *
 * Return Codes:
 *       E_SUCCESS                 - operation completed successfully
 *        EP_RB_RECOVER_BAD_ARGS
 *        EP_RB_RECOVER_RPC_FAIL
 *        EP_RB_RECOVER_SERVER_FAIL
 *        EP_RB_RECOVER_INVALOP           -If no question awaiting
                                                                    answer
 ********************************************************************/

eerrno_ty EDMRST_GetQuestion( serverHandle      svrHdl,
                              queryObjectPtr    queryPtr );

/*********************************************************************
 * SetUserAnswer
 *
 * This function is used to return user input requested via the
                                                                feedbackPtr
 * parameter output of the GetRestoreFeedback function call.
 *
 * Parameters:
 *
 * svrHdl   (I) - A pointer to this user's client handle for
 *                the Restore Engine (server) connection.
 * QueryPtr (
          I) - Pointer to structure containing the query and response
 *              data.
 * answer   (I) - pointer to text string response to question.
 * more      (
       I) - indicator that there will be more answers to this question
 *
 * Return Codes:
 *       E_SUCCESS                 - operation completed successfully
 *        EP_RB_RECOVER_BAD_ARGS
 *        EP_RB_RECOVER_NOMEM
 *        EP_RB_RECOVER_RPC_FAIL
 *        EP_RB_RECOVER_SERVER_FAIL
 *        EP_RB_RECOVER_INVALOP           -If no question awaiting
                                                                    answer
 *
 ********************************************************************/

eerrno_ty EDMRST_SetUserAnswer( serverHandle     svrHdl,
                                queryObjectPtr   queryPtr,
                                const char       *answer,
                                boolean_ty       more );
```



```
/*********************************************************************
 * Query Object Access Routines:
 *
 *  These routines retrieve portions of a specified Query object.
                                                                  A query
 *  object is returned by EDMRST_GetQuestion,
                                                 and is used to get a response
 *  from the Restore API 'operator' after a restore has been started.
 *
 * Parameters common to all these functions:
 *  svrHdl     (I) - A pointer to this user's client handle for the
 *                   Restore Engine (
                                          server) connection. Must be valid.
 *  thisObject (I) - The query object
 *
 * parameters specific to only one function:
 *  minlen     (O) - minumum response length
 *  maxlen     (O) - maximum response length
 *  isset      (
               O) - boolean_ty indicating this is the default response
 *  cookie    (IO) - placeholder in list of possible choices
 *
 * Returns one of the following:
 *  int                question type or number of possible responses;
                                                                  0 on errs
 *  const char *       ptr to a text string in the query object,
                                                                NULL on errs
 *  eerrno_ty          E_SUCCESS; or EP_RB_RECOVER_xxx on errors
 *
 *********************************************************************/

int EDMRST_GetQuestionType (serverHandle   svrHdl,
                            queryObjectPtr thisObject );

eerrno_ty EDMRST_GetQuestionAnswerSize (serverHandle    svrHdl,
                                        queryObjectPtr  thisObject,
                                        int             *minlen,
                                        int             *maxlen );

int EDMRST_GetQuestionNumChoices( serverHandle   svrHdl,
                                  queryObjectPtr thisObject );

const char * EDMRST_GetQuestionNextChoice(
                                          serverHandle         svrHdl,
                                          queryObjectPtr
                                                          thisObject,
                                          int                  *isset,
                                          long
                                                           *cookie );

const char * EDMRST_GetQuestionInvalidChars(
                                            serverHandle       svrHdl,
                                            queryObjectPtr
                                                        thisObject );

const char * EDMRST_GetQuestionHeaderText(
                                          serverHandle         svrHdl,
                                          queryObjectPtr
                                                        thisObject );

const char * EDMRST_GetQuestionText( serverHandle       svrHdl,
                                     queryObjectPtr     thisObject );
```

```
1575  /*****************************************************************/
1576  /* EDMRST_GetHostPlatformType                                    */
1577  /*                                                               */
1578  /*       This function returns the platform type of the          */
1579  /*       specified host.                                         */
                                                                      */
1580  /*                                                               */
1581  /* Parameters:                                                   */
1582  /*       svrHdl          -  (I) A pointer to this user's client  */
1583  /*                              handle for the Restore Engine    */
1584  /*                              connection.                      */
1585  /*       pHostName        - (I) ptr to hostname string;          */
1586  /*       pType            - (O) ptr to host platform type, which */
1587  /*                              can be one of the following valid*/
1588  /*                              types:                           */
1589  /*                                                               */
1590  /*                                BUCFG_PLATFORM_UNIX            */
1591  /*                                BUCFG_PLATFORM_OS2             */
1592  /*                                BUCFG_PLATFORM_WNT             */
1593  /*                                BUCFG_PLATFORM_NETWARE         */
1594  /*                                                               */
1595  /* Return Codes:                                                 */
1596  /*       E_SUCCESS:                                              */
1597  /*                         - operation successful, *pType will   */
1598  /*                           contain one of the valid types      */
1599  /*                           described above.                    */
1600  /*                                                               */
1601  /*        EP_RB_RECOVER_BAD_ARGS                                 */
1602  /*        EP_RB_RECOVER_RPC_FAIL                                 */
1603  /*        EP_RB_RECOVER_SERVER_FAIL                              */
1604  /*        EP_RB_RECOVER_INVALOP    - If another request active   */
1605  /*                                                               */
1606  /*****************************************************************/
1607  eerrno_ty
1608  EDMRST_GetHostPlatformType( serverHandle      svrHdl,
1609                              const char        *pHostName,
1610                              BuCfgPlatformType *pType );


1613  /*****************************************************************
1614   * EDMRST_GetBackupTimesSupport
1615   *
1616   * Function Description:
1617   *       Determine if the specified top level object can be restored
1618   *       from multiple backup times.
1619   *
1620   * Parameters:
1621   *       svrHdl       (I) - A pointer to this user's client handle for
1622   *                          the Restore Engine (server) connection.
1623   *       thisObject   (I) - The restoral object
1624   *       isSupported (
                 O) - TRUE/FALSE that restores from multiple backup planes
1625   *                          is/is not supported
1626   *
1627   * Return Codes:
1628   *       E_SUCCESS                 - operation completed successfully
1629   *       EP_RB_RECOVER_BAD_ARGS
1630   *       EP_RB_RECOVER_RPC_FAIL
1631   *       EP_RB_RECOVER_SERVER_FAIL
1632   *       EP_RB_RECOVER_INVALOP           -If another request active
1633   *
1634   *****************************************************************
                                                                      */

1636  eerrno_ty EDMRST_GetBackupTimesSupport(  serverHandle    svrHdl,
1637                                           restorableObjectPtr
```



```
                                                        thisObject,
1638                                        boolean_ty      *isSupported
                                                                    );

1640  /*****************************************************************/
1641  /* Query available backups                                       */
1642  /*                                                               */
1643  /*       There are 4 API functions in this group:                */
1644  /*               EDMRST_IsTherePrevBackup()                      */
1645  /*               EDMRST_IsThereNextBackup()                      */
1646  /*               EDMRST_IsTherePrevBackupForTime()               */
1647  /*               EDMRST_IsThereNextBackupForTime()               */
1648  /*                                                               */
1649  /*       These functions allow the caller to query whether or    */
1650  /*       not a previous or next backup exists relative to either */
1651  /*       the currently selected backup, or the backup of the     */
1652  /*       specified time. Additionally, the caller can specify    */
1653  /*       selction flags such as complete backups only.           */
1654  /*****************************************************************/

1656  /*****************************************************************
1657   * EDMRST_IsTherePrevBackup
1658   *
1659   * Function Description:
1660   *       Determine if a backup exists prior to the backup that is
1661   *       currently selected.
1662   *
1663   * Parameters:
1664   *       svrHdl  (I) - A pointer to this user's client handle for
1665   *                     the Restore Engine (server) connection.
1666   *       flags   (
                   I) - Backup constraint flags: e.g., full-only/partial-ok
1667   *       isThere  (O) - TRUE/FALSE that requested backup does exist
1668   *
1669   * Return Codes:
1670   *       E_SUCCESS                 - operation completed successfully
1671   *       EP_RB_RECOVER_BAD_ARGS
1672   *       EP_RB_RECOVER_RPC_FAIL
1673   *       EP_RB_RECOVER_SERVER_FAIL
1674   *       EP_RB_RECOVER_INVALOP           -If another request active
1675   *       EP_RB_RECOVER_NO_SAVESET        - when errors occur accessing
                                                                savesets
1676   *       EP_RB_RECOVER_NO_CATALOG        - when errors occur accessing
                                                                catalogs
1677   *
1678   *****************************************************************
                                                                      */


1680  eerrno_ty
1681  EDMRST_IsTherePrevBackup( serverHandle   svrHdl,
1682                            u_long                 flags,
1683                            boolean_ty     *isThere );

1685  /*****************************************************************
1686   * EDMRST_IsThereNextBackup
1687   *
1688   * Function Description:
1689   *       Determine if a backup exists after the backup that is
1690   *       currently selected.
1691   *
1692   * Parameters:
1693   *       svrHdl  (I) - A pointer to this user's client handle for
1694   *                     the Restore Engine (server) connection.
1695   *       flags   (
                   I) - Backup constraint flags: e.g., full-only/partial-ok
1696   *       isThere  (O) - TRUE/FALSE that requested backup does exist
```

```
 *
 * Return Codes:
 *       E_SUCCESS                  - operation completed successfully
 *        EP_RB_RECOVER_BAD_ARGS
 *        EP_RB_RECOVER_RPC_FAIL
 *        EP_RB_RECOVER_SERVER_FAIL
 *        EP_RB_RECOVER_INVALOP           -If another request active
 *       EP_RB_RECOVER_NO_SAVESET         - when errors occur accessing
                                                              savesets
 *       EP_RB_RECOVER_NO_CATALOG         - when errors occur accessing
                                                              catalogs
 *
 ************************************************************************
                                                                      */


eerrno_ty
EDMRST_IsThereNextBackup( serverHandle   svrHdl,
                          u_long         flags,
                          boolean_ty     *isThere );

/*****************************************************************
 * EDMRST_IsTherePrevBackupForTime
 *
 * Function Description:
 *      Determine if a backup exists prior to the specified time
 *
 * Parameters:
 *       svrHdl  (I) - A pointer to this user's client handle for
 *                     the Restore Engine (server) connection.
 *      thisTime(I) - Time for the query
 *      flags   (
                I) - Backup constraint flags: e.g., full-only/partial-ok
 *      isThere (O) - TRUE/FALSE that requested backup does exist
 *
 * Return Codes:
 *       E_SUCCESS                  - operation completed successfully
 *        EP_RB_RECOVER_BAD_ARGS
 *        EP_RB_RECOVER_RPC_FAIL
 *        EP_RB_RECOVER_SERVER_FAIL
 *        EP_RB_RECOVER_INVALOP           -If another request active
 *       EP_RB_RECOVER_NO_SAVESET         - when errors occur accessing
                                                              savesets
 *       EP_RB_RECOVER_NO_CATALOG         - when errors occur accessing
                                                              catalogs
 *
 ************************************************************************
                                                                      */


eerrno_ty
EDMRST_IsTherePrevBackupForTime( serverHandle           svrHdl,
                                 const time_t           thisTime,
                                 u_long                       flags,
                                 boolean_ty             *isThere );

/*****************************************************************
 * EDMRST_IsThereNextBackupForTime
 *
 * Function Description:
 *      Determine if a backup exists after to the specified time
 *
 * Parameters:
 *      context (I) - Pointer to the restore context
 *      thisTime(I) - Time for the query
 *      flags   (
                I) - Backup constraint flags: e.g., full-only/partial-ok
 *      isThere (O) - TRUE/FALSE that requested backup does exist
```



```
 *
 * Return Codes:
 *       E_SUCCESS                  - operation completed successfully
 *        EP_RB_RECOVER_BAD_ARGS
 *        EP_RB_RECOVER_RPC_FAIL
 *        EP_RB_RECOVER_SERVER_FAIL
 *        EP_RB_RECOVER_INVALOP           -If another request active
 *       EP_RB_RECOVER_NO_SAVESET         - when errors occur accessing
                                                              savesets
 *       EP_RB_RECOVER_NO_CATALOG         - when errors occur accessing
                                                              catalogs
 *
 ************************************************************************
                                                                      */


eerrno_ty
EDMRST_IsThereNextBackupForTime( serverHandle           svrHdl,
                                 const time_t           thisTime,
                                 u_long                 flags,
                                 boolean_ty             *isThere );

/*****************************************************************
 * Query WIProgress Access Routines:
 *
 *  These routines retrieve portions of a specified WIProgress object.
                                                                    A
 *  WIProgress object is returned by EDMRST_XXXXXXXX,
                                                  and is used to get
 *  the first WIProgress from a EDMStats objecti which is return from
                                                                  the
 *  EMMRST_XXXXXXXXX call.
 *
 * Parameters common to all these functions:
 *  svrHdl     (I) - A pointer to this user's client handle for the
 *                   Restore Engine (
                                  server) connection. Must be valid.
 *  thisObject (I) - The query object
 *
 * Returns one of the following:
 *
 *****************************************************************/

const char * EDMRST_GetWIObjectName(
                        serverHandle svrHdl, WIProgressPtr thisObject);
const char * EDMRST_GetWIObjectTrailName(
                        serverHandle svrHdl, WIProgressPtr thisObject);
const char * EDMRST_GetWIObjectTrailSetName(
                        serverHandle svrHdl, WIProgressPtr thisObject);
const char * EDMRST_GetWIObjectTemplateName(
                        serverHandle svrHdl, WIProgressPtr thisObject);
const char * EDMRST_GetWIObjectClientName(
                        serverHandle svrHdl, WIProgressPtr thisObject);
const char * EDMRST_GetWIObjectServerName(
                        serverHandle svrHdl, WIProgressPtr thisObject);
const char * EDMRST_GetWIObjectMediaType(
                       serverHandle  svrHdl, WIProgressPtr thisObject);
const char * EDMRST_GetWIObjectUserId(
                        serverHandle svrHdl, WIProgressPtr thisObject);
const char * EDMRST_GetWIObjectLevel(
                        serverHandle svrHdl, WIProgressPtr thisObject);
const char * EDMRST_GetWIObjectType(
                        serverHandle svrHdl, WIProgressPtr thisObject);
u_long EDMRST_GetObjectTimeStarted(
                        serverHandle svrHdl, WIProgressPtr thisObject);
u_long EDMRST_GetObjectCurrTime(
                        serverHandle svrHdl, WIProgressPtr thisObject);
```

```
1802  u_long EDMRST_GetObjectTotalKbytesSofar(
                          serverHandle svrHdl, WIProgressPtr thisObject);
1803  u_long EDMRST_GetObjectTotalFiles(
                          serverHandle svrHdl, WIProgressPtr thisObject);
1804  u_long EDMRST_GetObjectTotalBadFiles(
                          serverHandle svrHdl, WIProgressPtr thisObject);
1805  u_long EDMRST_GetObjectCurrKbytesSofar(
                          serverHandle svrHdl, WIProgressPtr thisObject);
1806  u_long EDMRST_GetObjectCurrTimeSlice(
                          serverHandle svrHdl, WIProgressPtr thisObject);
1807  u_long EDMRST_GetObjectCurrFiles(
                          serverHandle svrHdl, WIProgressPtr thisObject);
1808  u_long EDMRST_GetObjectTotalFilesExpected(
                          serverHandle svrHdl, WIProgressPtr thisObject);
1809  u_long EDMRST_GetObjectTotalKbExpected(
                          serverHandle svrHdl, WIProgressPtr thisObject);
1810  u_long EDMRST_GetObjectItemStatus(
                          serverHandle svrHdl, WIProgressPtr thisObject);
1811  int EDMRST_GetObjectOperationType(
                          serverHandle svrHdl, WIProgressPtr thisObject);
1812  int EDMRST_GetObjectCompleted(
                          serverHandle svrHdl, WIProgressPtr thisObject);
1813  int EDMRST_IsLastObject(serverHandle svrHdl, WIProgressPtr thisObject);
1814  WIProgressPtr EDMRST_GetNextObject(
                          serverHandle svrHdl, WIProgressPtr thisObject);
1815  WIProgressPtr EDMRST_GetFirstObject(
                        serverHandle svrHdl,feedbackObjectPtr thisObject);


1817  EDMProgressPtr
1818  EDMRST_GetFirstEDMObject(
                       serverHandle svrHdl, feedbackObjectPtr thisObject);
1819  EDMProgressPtr
1820  EDMRST_GetNextEDMObject(
                         serverHandle svrHdl, EDMProgressPtr thisObject);
1821  u_long
1822  EDMRST_GetObjectEDMTimeStarted(
                        serverHandle svrHdl, EDMProgressPtr thisObject);
1823  u_long
1824  EDMRST_GetObjectEDMCurrTime(
                          serverHandle svrHdl,EDMProgressPtr thisObject);
1825  u_long
1826  EDMRST_GetObjectEDMTotalKbytesSofar(
                          serverHandle svrHdl,EDMProgressPtr thisObject);
1827  u_long
1828  EDMRST_GetObjectEDMTotalFiles(
                          serverHandle svrHdl,EDMProgressPtr thisObject);
1829  u_long
1830  EDMRST_GetObjectEDMTotalBadFiles(
                          serverHandle svrHdl,EDMProgressPtr thisObject);
1831  u_long
1832  EDMRST_GetObjectEDMCurrTimeSlice(
                          serverHandle svrHdl,EDMProgressPtr thisObject);
1833  u_long
1834  EDMRST_GetObjectEDMCurrKbytesSofar(
                          serverHandle svrHdl,EDMProgressPtr thisObject);
1835  u_long
1836  EDMRST_GetObjectEDMCurrFiles(
                          serverHandle svrHdl,EDMProgressPtr thisObject);
1837  u_long
1838  EDMRST_GetObjectEDMActive(
                          serverHandle svrHdl,EDMProgressPtr thisObject);
1839  u_long
1840  EDMRST_GetObjectEDMTotal(
                          serverHandle svrHdl,EDMProgressPtr thisObject);
1841  u_long
1842  EDMRST_GetObjectEDMFailed(
```



```
                          serverHandle svrHdl,EDMProgressPtr thisObject);
1843  u_long
1844  EDMRST_GetObjectEDMSuccessful(
                          serverHandle svrHdl,EDMProgressPtr thisObject);
1845  u_long
1846  EDMRST_GetObjectEDMTotalFilesExpected(
                          serverHandle svrHdl,EDMProgressPtr thisObject);
1847  u_long
1848  EDMRST_GetObjectEDMTotalKBExpected(
                          serverHandle svrHdl,EDMProgressPtr thisObject);
1849  int
1850  EDMRST_GetObjectEDMOperationType(
                          serverHandle svrHdl,EDMProgressPtr thisObject);
1851  int
1852  EDMRST_GetObjectEDMCompleted(
                          serverHandle svrHdl,EDMProgressPtr thisObject);
1853  u_long
1854  EDMRST_GetObjectEDMStatus(
                          serverHandle svrHdl,EDMProgressPtr thisObject);
1855  const char *
1856  EDMRST_GetObjectEDMHostName(
                          serverHandle svrHdl,EDMProgressPtr thisObject);
1857  /** Notify Access Routines */
1858  void*
1859  EDMRST_GetFirstNotifyObject(
                       serverHandle svrHdl,feedbackObjectPtr thisObject);
1860  void*
1861  EDMRST_GetNextNotifyObject(serverHandle svrHdl,void* thisObject);
1862  const char *
1863  EDMRST_GetNofifyMessageText(serverHandle  svrHdl,void* thisObject);
1864  int
1865  EDMRST_GetNotifyMessageType(serverHandle svrHdl,void* thisObject);
1866  int
1867  EDMRST_GetNotifySourceModule(serverHandle svrHdl,void* thisObject);
1868  int
1869  EDMRST_GetNotifyLevel(serverHandle svrHdl,void* thisObject);
1870  int
1871  EDMRST_GetNotifyMessageLength(serverHandle svrHdl,void* thisObject);
1872  int
1873  EDMRST_IsLastNotifyObject(serverHandle svrHdl,void* thisObject);



1876  /*************************************************************************
1877   * SetRecxDirectives:
1878   *
1879   *  This routine returns sends the Filename and path plus hostname
1880   *    of the recx directives file, which was created by the command
1881   *    eb_dc_restore, to the server which then processes the recx
1882   *     directives
1883   *
1884   * Parameters:
1885   *  svrHdl    (I) - A pointer to this user's client handle for the
1886   *                  Restore Engine (server) connection.
1887   *  template  (O) - The name of the local recx file
1888   *  alternate (O) - the name of this host so the file can be tranfered
1889   *
1890   *************************************************************************/
1891  eerrno_ty EDMRST_SetRecxDirectives( serverHandle     svrHdl,
1892                                      char             *filename,
1893                                     char             *hostname );



1896  /*************************************************************************
1897   * EDMRST_get_catalog_info:
1898   *
1899   *  This routine returns sends the fills the level string with the
```

```
1900    *  level for backup being restored
1901    *
1902    * Parameters:
1903    *  svrHdl           (
                          I) - A pointer to this user's client handle for the
1904    *                        Restore Engine (server) connection.
1905    *  backup_time      (I) - Time of the backup that is beeing looked at
1906    *  *level           (O) - The level of the backup for specified time
1907    *                        taken from catalog structure.  If not enough
1908    *                        memory has been allocated value will be "\0"
1909    *  *numrec          (
                           O) - The number of records for the specified backup
1910    *                        taken from catalog structure.   If not enough
1911    *                        memory has been allocated value will be "\0"
1912    *  *catType         (O) - The type of catalog for the specified bacup
1913    *                        taken from catalog structure.   If not enough
1914    *                        memory has been allocated value will be "\0"
1915    * Return Codes:
1916    *   EP_RB_RECOVER_BAD_ARGS - arguments passed in are null
1917    *   E_SUCCESS              - the fields have been filled in
1918    *                            and RPC succeeded
1919    *
1920    ************************************************************************/
1921   eerrno_ty
1922   EDMRST_getCatalogInfo(   serverHandle   svrHdl,
1923                            time_t          backup_time,
1924                            char            *level,
1925                            char            *numRec,
1926                            char            *catType);

1928   #endif   /* H_RESTOREAPI */
```

```
/* -- Template created by NEURON DATA Open Interface.                    */
/* -- Do not alter 'CodeGen' directives.                                 */
/* (( CodeGen: GeneratorVersion 4 ))                                     */

/* -- Code generated on 05/20/97 at 16:10:09.                            */
/* -- Code regenerated on 05/22/97 at 16:07:31.                          */
/* -- Code regenerated on 05/22/97 at 16:09:45.                          */
/* -- Code regenerated on 05/23/97 at 09:24:06.                          */
/* -- Code regenerated on 05/23/97 at 09:34:04.                          */
/* -- Code regenerated on 05/23/97 at 09:34:48.                          */
/* -- Code regenerated on 09/22/97 at 12:51:42.                          */
/* -- Code regenerated on 09/22/97 at 16:36:51.                          */
/* -- Code regenerated on 03/18/98 at 11:14:28. */
/* -- Code regenerated on 05/18/99 at 10:10:12.                          */
/* -- Code regenerated on 05/18/99 at 10:54:55.                          */
/* -- Code regenerated on 06/18/99 at 15:20:38.                          */
/* -- Code regenerated on 07/15/99 at 11:19:43.                          */
/* -- Code regenerated on 07/16/99 at 15:49:52.                          */
/* -- Code regenerated on 09/03/99 at 10:48:06.                          */
/* (( CodeGen: CodeHistory ))                                            */

#define ERR_LIB RESTORE

#include <stdlib.h>
#include <libgen.h>

#include <esl/c_portable.h>

#define REST_INIT
#include "restoreP.h"
#include "restore.h"
#undef REST_INIT

#include <X11/Xlib.h>
#include <xpub.h>

#include "restore/restMgr.h"
#include "restSearch.h"
#include "restSelMgr.h"
#include "restCBMgr.h"
#include "restUtils.h"
#include "gutil/aboutMgr.h"
#include "gutil/alertMgr.h"
#include "gutil/grpMgr.h"
#include "gutil/guiDefines.h"
#include "gutil/guiUtils.h"
#include "gutil/miscUtils.h"
#include "gutil/iconDefs.h"
```



```
#include "gutil/iconUtils.h"
#include "gutil/hostUtils.h"
#include "gutil/winUtils.h"
#include "gutil/tedUtils.h"
#include "gutil/lboxUtils.h"
#include "gutil/cboxUtils.h"
#include "gutil/codeTracer.h"
#include "gutil/resUtils.h"
#include "help/helpIpc.h"
#include "ipc/ipcl.h"

ERR_EXTERN
ERR_INMODULE("restore")

#define  HELP_OPTION    "help"
#define  VERSION_OPTION "v"
#define  SYNC_OPTION    "sync"

/* (( CodeGen: ClassImplementationPlaceHolder ))                         */
/* (( CodeGen: WinClassImplementationPlaceHolder ))                      */

/* (( CodeGen: WindowSection RestoreWin                                  */
/* ======================================================================= */
/* == Code for Window "RestoreWin"                                       */
/* ======================================================================= */

/* (( CodeGen: MenuImplementationPlaceHolder ))                          */

static  void  C_FAR  S_RestoreWinNfy L2(WinPtr, win, WinNfyEnum, code)
{

  /* Call the restore notify routine so update the timeout status */
  REST_NotifyReceived (code);

  switch (code) {
  case WIN_NFYTERMINATE:
    if (REST_Remove ())
    {
      WIN_DefNfy(win, code);
      EVENT_MainExit ();
    }
    break;
  /* USER CODE */
  case WIN_NFYMOUSECLICK:
    GUTIL_WinHandleMouseClick (win);
    break;
  case WIN_NFYREDRAW:
    WIN_DefNfy(win, code);
    /*
     * Update the focus borders
     */

    if (WGT_HasFocus ((
                       WgtPtr)REST_RestoreWin->BackupListBox, BOOL_TRUE))
      GUTIL_DrawFocusBorder ((
                       WgtPtr)REST_RestoreWin->BackupListBox, BOOL_TRUE);

    /* Use the junkTed to determine if the backup sarea has focus */
    if (WGT_HasFocus ((WgtPtr)REST_RestoreWin->JunkTEd, BOOL_TRUE))
```

```
106 2         GUTIL_DrawFocusBorder ((
                          WgtPtr)REST_RestoreWin->BackupSArea, BOOL_TRUE);

108 2       if (WGT_HasFocus ((
                        WgtPtr)REST_RestoreWin->SelectedListBox, BOOL_TRUE))
109 2         GUTIL_DrawFocusBorder ((
                        WgtPtr)REST_RestoreWin->SelectedListBox, BOOL_TRUE);

111 2       if (WGT_HasFocus ((
                          WgtPtr)REST_RestoreWin->MediaListBox, BOOL_TRUE))
112 2         GUTIL_DrawFocusBorder ((
                          WgtPtr)REST_RestoreWin->MediaListBox, BOOL_TRUE);

114 2       break;
115 2     case WIN_NFYRESIZE:
116 2       GUTIL_WinHandleResize ((WinPtr)win);
117 2       break;

119 2     default:
120 2       WIN_DefNfy(win, code);
121 1     }

123     }
```



```
125   static  void  C_FAR  S_MediaListBoxNfy L2(
                                              LBoxPtr, lbox, LBoxNfyEnum, code)
126 1 {

128 2     switch (code) {
129 2     /* USER CODE */
130 2     case LBOX_NFYCELLDELETE:
131 2       REST_DisposeMediaInfo (lbox);
132 2       break;
133 2     default:
134 2       GUTIL_LBOX_DefNfy (lbox, code, REST_GetMediaListColumnValues);
135 1     }

137   }
```

```
139     static  void  C_FAR  S_SelectedListBoxNfy L2(
                                        LBoxPtr, lbox, LBoxNfyEnum, code)
140   1 {

142   2   switch (code) {
143   2   /* USER CODE */
144   2   case LBOX_NFYCELLDELETE:
145   2     REST_DisposeSelectedInfo (lbox);
146   2     break;
147   2   case LBOX_NFYSELOPERATION:
148   2     REST_UpdateRemoveButtons ();
149   2     break;
150   2   default:
151   2     GUTIL_LBOX_DefNfy(lbox, code, REST_GetSelectedListColumnValues);
152   1   }

154     }
```



```
156     /* (( CodeGen: WgtNfyHandler HitPrevBackupButton
                                                                    */
157     static   void C_FAR RestoreRestoreWin_HitPrevBackupButton L1(
                                                RestoreRestoreWinPtr, win)
158   1 {
159   1   REST_PrevButtonSelect ();
160     }
```

```
161   /* )) CodeGen: WgtNfyHandler HitPrevBackupButton
                                                                */


163   /* (( CodeGen: WgtNfyHandler HitCalendarButton
                                                                */
164   static   void C_FAR RestoreRestoreWin_HitCalendarButton L1(
                                                  RestoreRestoreWinPtr, win)
165 1 {
166 1   REST_CalendarButtonSelect ();
167   }
```



```
168   /* )) CodeGen: WgtNfyHandler HitCalendarButton
                                                                */


170   /* (( CodeGen: WgtNfyHandler HitNextBackupButton
                                                                */
171   static   void C_FAR RestoreRestoreWin_HitNextBackupButton L1(
                                                  RestoreRestoreWinPtr, win)
172 1 {
173 1   REST_NextButtonSelect ();
174   }
```

```
175   /* )) CodeGen: WgtNfyHandler HitNextBackupButton
                                                        */

177   /* (( CodeGen: WgtNfyHandler EltSelectedTemplateBox
                                                        */
178   static   void C_FAR RestoreRestoreWin_EltSelectedTemplateBox L2(
                    RestoreRestoreWinPtr, win, CBoxEltSelectedNfyCPtr, info)
179 1 {
180 1   REST_UpdateTemplateFromBoxes ();
181   }
```



```
182   /* )) CodeGen: WgtNfyHandler EltSelectedTemplateBox
                                                        */

184   static BoolEnum S_TemplateSelectProc (Str selectedStr)
185 1 {
186 1   REST_UpdateTemplateFromBoxes ();

188 1   return (BOOL_TRUE);
189   }
```

```
191   static  void  C_FAR  S_TemplateBoxNfy L2(
                                    CBoxPtr, cbox, CBoxNfyEnum, code)
192 1 {
193 1     GUTIL_CBOX_DefNfy (cbox, code, S_TemplateSelectProc);
194   }
```



```
196   /* (( CodeGen: WgtNfyHandler EltSelectedPrimaryBox
                                                          */
197   static   void C_FAR RestoreRestoreWin_EltSelectedPrimaryBox L2(
                RestoreRestoreWinPtr, win, CBoxEltSelectedNfyCPtr, info)
198 1 {
199 1   REST_UpdateTemplateFromBoxes ();
200   }
```

```
201   /* )) CodeGen: WgtNfyHandler EltSelectedPrimaryBox
                                                                */
203   /* (( CodeGen: WgtNfyHandler ValidateJunkTEd
                                                                */
204   static   void C_FAR RestoreRestoreWin_ValidateJunkTEd L1(
                                                  RestoreRestoreWinPtr, win)
205 1 {
206   }
```



```
207   /* )) CodeGen: WgtNfyHandler ValidateJunkTEd
                                                                */
209   /* (( CodeGen: WgtNfyHandler CellStringBackupListBox
                                                                */
210   static   void C_FAR RestoreRestoreWin_CellStringBackupListBox L2(
                                  RestoreRestoreWinPtr, win, LBoxStringPtr, lbs)
211 1 {
212   }
```

```
213   /* )) CodeGen: WgtNfyHandler CellStringBackupListBox
                                                          */

215   /* (( CodeGen: WgtNfyHandler ValidateBackupListBox
                                                          */
216   static   void C_FAR RestoreRestoreWin_ValidateBackupListBox L1(
                                                  RestoreRestoreWinPtr, win)
217 1 {
218   }
```



```
219   /* )) CodeGen: WgtNfyHandler ValidateBackupListBox
                                                          */

221   /* (( CodeGen: WgtNfyHandler HitSearchButton
                                                          */
222   static   void C_FAR RestoreRestoreWin_HitSearchButton L1(
                                                  RestoreRestoreWinPtr, win)
223 1 {
224 1   REST_DisplaySearch ();
225   }
```

```
226   /* )) CodeGen: WgtNfyHandler HitSearchButton
                                                                    */

228   /* (( CodeGen: WgtNfyHandler HitMarkButton
                                                                    */
229   static   void C_FAR RestoreRestoreWin_HitMarkButton L1(
                                                  RestoreRestoreWinPtr, win)
230 1 {
231 1   REST_MarkBackupItems ();
232   }
```



```
233   /* )) CodeGen: WgtNfyHandler HitMarkButton
                                                                    */

235   /* (( CodeGen: WgtNfyHandler HitUnmarkButton
                                                                    */
236   static   void C_FAR RestoreRestoreWin_HitUnmarkButton L1(
                                                  RestoreRestoreWinPtr, win)
237 1 {
238 1   REST_UnmarkBackupItems ();
239   }
```

```
240   /* )) CodeGen: WgtNfyHandler HitUnmarkButton
                                                                */


242   /* (( CodeGen: WgtNfyHandler HitTypeSortButton
                                                                */
243   static   void C_FAR RestoreRestoreWin_HitTypeSortButton L1(
                                                  RestoreRestoreWinPtr, win)
244 1 {
245 1   REST_SetSort (REST_ByType);
246   }
```



```
247   /* )) CodeGen: WgtNfyHandler HitTypeSortButton
                                                                */


249   /* (( CodeGen: WgtNfyHandler HitNameSortButton
                                                                */
250   static   void C_FAR RestoreRestoreWin_HitNameSortButton L1(
                                                  RestoreRestoreWinPtr, win)
251 1 {
252 1   REST_SetSort (REST_ByName);
253   }
```

```
254   /* )) CodeGen: WgtNfyHandler HitNameSortButton
                                                                    */

256   /* (( CodeGen: WgtNfyHandler HitOwnerSortButton
                                                                    */
257   static   void C_FAR RestoreRestoreWin_HitOwnerSortButton L1(
                                                  RestoreRestoreWinPtr, win)
258 1 {
259 1   REST_SetSort (REST_ByOwner);
260   }
```



```
261   /* )) CodeGen: WgtNfyHandler HitOwnerSortButton
                                                                    */

263   /* (( CodeGen: WgtNfyHandler HitSizeSortButton
                                                                    */
264   static   void C_FAR RestoreRestoreWin_HitSizeSortButton L1(
                                                  RestoreRestoreWinPtr, win)
265 1 {
266 1   REST_SetSort (REST_BySize);
267   }
```

```
268   /* )) CodeGen: WgtNfyHandler HitSizeSortButton
                                                                */


270   /* (( CodeGen: WgtNfyHandler HitDateSortButton
                                                                */
271   static   void C_FAR RestoreRestoreWin_HitDateSortButton L1(
                                                  RestoreRestoreWinPtr, win)
272 1 {
273 1   REST_SetSort (REST_ByDate);
274   }
```



```
275   /* )) CodeGen: WgtNfyHandler HitDateSortButton
                                                                */


277   /* (( CodeGen: WgtNfyHandler HitHiddenButton
                                                                */
278   static   void C_FAR RestoreRestoreWin_HitHiddenButton L1(
                                                  RestoreRestoreWinPtr, win)
279 1 {
280 1   REST_ShowHiddenFiles = TBUT_GetSelected ((TButPtr)win->HiddenButton);
281 1   REST_UpdateViewOptions ();
282   }
```

```
283   /* )) CodeGen: WgtNfyHandler HitHiddenButton
                                                                    */
285   /* (( CodeGen: WgtNfyHandler HitBadFilesButton
                                                                    */
286   static   void C_FAR RestoreRestoreWin_HitBadFilesButton L1(
                                                  RestoreRestoreWinPtr, win)
287 1 {
288 1   REST_ShowBadFiles = TBUT_GetSelected ((TButPtr)win->BadFilesButton);
289 1   REST_UpdateViewOptions ();
290   }
```



```
291   /* )) CodeGen: WgtNfyHandler HitBadFilesButton
                                                                    */
293   /* (( CodeGen: WgtNfyHandler HitMarkBadButton
                                                                    */
294   static   void C_FAR RestoreRestoreWin_HitMarkBadButton L1(
                                                  RestoreRestoreWinPtr, win)
295 1 {
296 1   REST_MarkBadFiles = TBUT_GetSelected ((TButPtr)win->MarkBadButton);
297   }
```

```
298   /* )) CodeGen: WgtNfyHandler HitMarkBadButton
                                                                */


300   /* (( CodeGen: WgtNfyHandler ValidateRestoreItemsTarea
                                                                */
301   static   void C_FAR RestoreRestoreWin_ValidateRestoreItemsTarea L1(
                                                    RestoreRestoreWinPtr, win)
302 1 {
303   }
```



```
304   /* )) CodeGen: WgtNfyHandler ValidateRestoreItemsTarea
                                                                */


306   /* (( CodeGen: WgtNfyHandler ValidateRestoreSizeTarea
                                                                */
307   static   void C_FAR RestoreRestoreWin_ValidateRestoreSizeTarea L1(
                                                    RestoreRestoreWinPtr, win)
308 1 {
309   }
```

```
310   /* )) CodeGen: WgtNfyHandler ValidateRestoreSizeTarea
                                                                */
312   /* (( CodeGen: WgtNfyHandler ValidateBadFilesText
                                                                */
313   static   void C_FAR RestoreRestoreWin_ValidateBadFilesText L1(
                                                  RestoreRestoreWinPtr, win)
314 1 {
315   }
```



```
316   /* )) CodeGen: WgtNfyHandler ValidateBadFilesText
                                                                */
318   /* (( CodeGen: WgtNfyHandler HitRemoveButton
                                                                */
319   static   void C_FAR RestoreRestoreWin_HitRemoveButton L1(
                                                  RestoreRestoreWinPtr, win)
320 1 {
321 1   REST_RemoveSelectedItems ();
322   }
```

```
323   /* )) CodeGen: WgtNfyHandler HitRemoveButton
                                                                    */

325   /* (( CodeGen: WgtNfyHandler HitClearButton
                                                                    */
326   static   void C_FAR RestoreRestoreWin_HitClearButton L1(
                                                  RestoreRestoreWinPtr, win)
327 1 {
328 1   REST_RemoveAllSelectedItems ();
329   }
```



```
330   /* )) CodeGen: WgtNfyHandler HitClearButton
                                                                    */

332   /* (( CodeGen: WgtNfyHandler HitCloseButton
                                                                    */
333   static   void C_FAR RestoreRestoreWin_HitCloseButton L1(
                                                  RestoreRestoreWinPtr, win)
334 1 {
335 1   WIN_Terminate ((WinPtr)win);
336   }
```

```
337   /* )) CodeGen: WgtNfyHandler HitCloseButton
                                                                    */

339   /* (( CodeGen: WgtNfyHandler HitStartButton
                                                                    */
340   static   void C_FAR RestoreRestoreWin_HitStartButton L1(
                                                  RestoreRestoreWinPtr, win)
341 1 {
342 1   REST_StartRestoral ();
343   }
```



```
344   /* )) CodeGen: WgtNfyHandler HitStartButton
                                                                    */

346   static WgtPtr focusWgt;

348   static   void C_FAR RestoreRestoreWin_FocusHelpButton L1(
                                                  RestoreRestoreWinPtr, win)
349 1 {
350 1   focusWgt = PANEL_GetFocusWgt ((PanelPtr)WGT_GetWin((WgtPtr)(
                                                  win->HelpButton)));
351   }
```

```
353  /* (( CodeGen: WgtNfyHandler HitHelpButton
                                                                    */
354  static   void C_FAR RestoreRestoreWin_HitHelpButton L1(
                                                RestoreRestoreWinPtr, win)
355  1 {
356  1   REST_DisplayContextHelp (focusWgt);
357    }
```



```
358  /* )) CodeGen: WgtNfyHandler HitHelpButton
                                                                    */

360  /* (( CodeGen: WgtNfyHandler ValidateBackupDateText
                                                                    */
361  static   void C_FAR RestoreRestoreWin_ValidateBackupDateText L1(
                                                RestoreRestoreWinPtr, win)
362  1 {
363    }
```

```
364   /* )) CodeGen: WgtNfyHandler ValidateBackupDateText
                                                          */
366   /* (( CodeGen: WgtNfyHandler HitViewOptionsTab
                                                          */
367   static   void C_FAR RestoreRestoreWin_HitViewOptionsTab L1(
                                                RestoreRestoreWinPtr, win)
368 1 {
369   }
```



```
370   /* )) CodeGen: WgtNfyHandler HitViewOptionsTab
                                                          */
372   /* (( CodeGen: WgtNfyHandler HitMarkSummaryTab
                                                          */
373   static   void C_FAR RestoreRestoreWin_HitMarkSummaryTab L1(
                                                RestoreRestoreWinPtr, win)
374 1 {
375   }
```

```
376   /* )) CodeGen: WgtNfyHandler HitMarkSummaryTab
                                                                */


378   /* (( CodeGen: WgtNfyHandler CellStringSelectedListBox
                                                                */
379   static   void C_FAR RestoreRestoreWin_CellStringSelectedListBox L2(
                            RestoreRestoreWinPtr, win, LBoxStringPtr, lbs)
380 1 {
381   }
```



```
382   /* )) CodeGen: WgtNfyHandler CellStringSelectedListBox
                                                                */


384   /* (( CodeGen: WgtNfyHandler ValidateSelectedListBox
                                                                */
385   static   void C_FAR RestoreRestoreWin_ValidateSelectedListBox L1(
                                                    RestoreRestoreWinPtr, win)
386 1 {
387   }
```

```
388  /* )) CodeGen: WgtNfyHandler ValidateSelectedListBox
                                                                */

390  /* (( CodeGen: WgtNfyHandler HitMediaTab
                                                                */
391  static   void C_FAR RestoreRestoreWin_HitMediaTab L1(
                                                RestoreRestoreWinPtr, win)
392  1 {
393    }
```



```
394  /* )) CodeGen: WgtNfyHandler HitMediaTab
                                                                */

396  /* (( CodeGen: WgtNfyHandler CellStringMediaListBox
                                                                */
397  static   void C_FAR RestoreRestoreWin_CellStringMediaListBox L2(
                                RestoreRestoreWinPtr, win, LBoxStringPtr, lbs)
398  1 {
399    }
```

```
400   /* )) CodeGen: WgtNfyHandler CellStringMediaListBox
                                                          */


402   /* (( CodeGen: WgtNfyHandler ValidateMediaListBox
                                                          */
403   static   void C_FAR RestoreRestoreWin_ValidateMediaListBox L1(
                                                  RestoreRestoreWinPtr, win)
404 1 {
405   }
```



```
406   /* )) CodeGen: WgtNfyHandler ValidateMediaListBox
                                                          */


408   /* (( CodeGen: WgtNfyHandler ValidatePathTEd
                                                          */
409   static   void C_FAR RestoreRestoreWin_ValidatePathTEd L1(
                                                  RestoreRestoreWinPtr, win)
410 1 {
411   }
```

```
412   /* )) CodeGen: WgtNfyHandler ValidatePathTEd
                                                                    */
414   /* (( CodeGen: WgtNfyHandler HitAllowPartialButton
                                                                    */
415   static   void C_FAR RestoreRestoreWin_HitAllowPartialButton L1(
                                                  RestoreRestoreWinPtr, win)
416 1 {
417   }
```



```
418   /* )) CodeGen: WgtNfyHandler HitAllowPartialButton
                                                                    */
419   /* (( CodeGen: WgtNfyPlaceHolder ))
                                                                    */
421   static  void  C_FAR  S_PathTEdNfy L2(TEdPtr, ted, TEdNfyEnum, code)
422 1 {
423 1   /* If this is a keyboard character, and it is a return,
                                                        set the view */
424 1   if ((code == TED_NFYKEYCHAR) &&
425 1       ((EVENT_GetKeyCode() == EVENT_KEYRETURN) ||
426 1        (EVENT_GetKeyCode() == EVENT_KEYENTER)))
427 2   {
428 2     REST_SetViewToPath ();
429 1   }
430 1   /* Else, let the utilities do their thing */
431 1   else
432 2   {
433 2     GUTIL_TED_DefNfy (ted, code);
434 1   }
435   }
```

```
437   /* (( CodeGen: UseDefaultNfyHandler name_of_nfy_handler ))
                                                                */
438   /* (( CodeGen: UseAllDefaultNfyHandlers name_of_wgt_member ))
                                                                */

440   void      RestoreRestoreWin_Construct L1(RestoreRestoreWinPtr, win)
441 1 {
442 1         /* ((
          CodeGen: WgtInitializations                           */
443 1         win->DataAvailablePanel = (PanelPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "DataAvailablePanel");
444 1         win->FSOptionsPanel = (PanelPtr)PANEL_GetNamedWgt((
                                          PanelPtr)win, "FSOptionsPanel");
445 1         win->BackupDateTarea = (TAreaPtr)PANEL_GetNamedWgt((
                                         PanelPtr)win, "BackupDateTarea");
446 1         win->BackupDateText = (STEdPtr)PANEL_GetNamedWgt((
                                          PanelPtr)win, "BackupDateText");
447 1         win->PrevBackupButton = (PButPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "PrevBackupButton");
448 1         win->CalendarButton = (PButPtr)PANEL_GetNamedWgt((
                                          PanelPtr)win, "CalendarButton");
449 1         win->NextBackupButton = (PButPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "NextBackupButton");
450 1         win->TemplateBox = (CBoxPtr)PANEL_GetNamedWgt((
                                           PanelPtr)win, "TemplateBox");
451 1         win->PrimaryBox = (CBoxPtr)PANEL_GetNamedWgt((
                                            PanelPtr)win, "PrimaryBox");
452 1         win->JunkTEd = (STEdPtr)PANEL_GetNamedWgt((
                                              PanelPtr)win, "JunkTEd");
453 1         win->BackupSArea = (SAreaPtr)PANEL_GetNamedWgt((
                                           PanelPtr)win, "BackupSArea");
454 1         win->BackupListBox = (LBoxPtr)PANEL_GetNamedWgt((
                                         PanelPtr)win, "BackupListBox");
455 1         win->LboxVSb = (SbPtr)PANEL_GetNamedWgt((
                                           PanelPtr)win, "LboxVSb");
456 1         win->PathTArea = (TAreaPtr)PANEL_GetNamedWgt((
                                          PanelPtr)win, "PathTArea");
457 1         win->PathTEd = (STEdPtr)PANEL_GetNamedWgt((
                                           PanelPtr)win, "PathTEd");
458 1         win->SearchButton = (PButPtr)PANEL_GetNamedWgt((
                                         PanelPtr)win, "SearchButton");
459 1         win->MarkButton = (PButPtr)PANEL_GetNamedWgt((
                                           PanelPtr)win, "MarkButton");
460 1         win->UnmarkButton = (PButPtr)PANEL_GetNamedWgt((
                                         PanelPtr)win, "UnmarkButton");
461 1         win->TabsPanel = (PanelPtr)PANEL_GetNamedWgt((
                                           PanelPtr)win, "TabsPanel");
462 1         win->MarkSummaryTab = (PButPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "MarkSummaryTab");
463 1         win->MarkSummaryPanel = (PanelPtr)PANEL_GetNamedWgt((
                                       PanelPtr)win, "MarkSummaryPanel");
464 1         win->ItemsSelectedLabel = (TAreaPtr)PANEL_GetNamedWgt((
                                      PanelPtr)win, "ItemsSelectedLabel");
465 1         win->SelectedListBox = (LBoxPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "SelectedListBox");
466 1         win->RestoreItemsLabel = (TAreaPtr)PANEL_GetNamedWgt((
                                       PanelPtr)win, "RestoreItemsLabel");
467 1         win->RestoreItemsTarea = (STEdPtr)PANEL_GetNamedWgt((
                                       PanelPtr)win, "RestoreItemsTarea");
468 1         win->RestoreItemsLabel1 = (TAreaPtr)PANEL_GetNamedWgt((
                                      PanelPtr)win, "RestoreItemsLabel1");
469 1         win->RestoreSizeTarea = (STEdPtr)PANEL_GetNamedWgt((
                                       PanelPtr)win, "RestoreSizeTarea");
470 1         win->BadFilesLabel = (TAreaPtr)PANEL_GetNamedWgt((
                                         PanelPtr)win, "BadFilesLabel");
```



```
471 1         win->BadFilesText = (STEdPtr)PANEL_GetNamedWgt((
                                          PanelPtr)win, "BadFilesText");
472 1         win->RemoveButton = (PButPtr)PANEL_GetNamedWgt((
                                          PanelPtr)win, "RemoveButton");
473 1         win->ClearButton = (PButPtr)PANEL_GetNamedWgt((
                                           PanelPtr)win, "ClearButton");
474 1         win->MediaTab = (PButPtr)PANEL_GetNamedWgt((
                                             PanelPtr)win, "MediaTab");
475 1         win->MediaPanel = (PanelPtr)PANEL_GetNamedWgt((
                                            PanelPtr)win, "MediaPanel");
476 1         win->MediaListBox = (LBoxPtr)PANEL_GetNamedWgt((
                                          PanelPtr)win, "MediaListBox");
477 1         win->ViewOptionsTab = (PButPtr)PANEL_GetNamedWgt((
                                         PanelPtr)win, "ViewOptionsTab");
478 1         win->ViewOptionsPanel = (PanelPtr)PANEL_GetNamedWgt((
                                       PanelPtr)win, "ViewOptionsPanel");
479 1         win->SortPanel = (PanelPtr)PANEL_GetNamedWgt((
                                            PanelPtr)win, "SortPanel");
480 1         win->TypeSortButton = (RButPtr)PANEL_GetNamedWgt((
                                         PanelPtr)win, "TypeSortButton");
481 1         win->NameSortButton = (RButPtr)PANEL_GetNamedWgt((
                                         PanelPtr)win, "NameSortButton");
482 1         win->OwnerSortButton = (RButPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "OwnerSortButton");
483 1         win->SizeSortButton = (RButPtr)PANEL_GetNamedWgt((
                                         PanelPtr)win, "SizeSortButton");
484 1         win->DateSortButton = (RButPtr)PANEL_GetNamedWgt((
                                         PanelPtr)win, "DateSortButton");
485 1         win->OptionsPanel = (PanelPtr)PANEL_GetNamedWgt((
                                          PanelPtr)win, "OptionsPanel");
486 1         win->HiddenButton = (CButPtr)PANEL_GetNamedWgt((
                                         PanelPtr)win, "HiddenButton");
487 1         win->BadFilesButton = (CButPtr)PANEL_GetNamedWgt((
                                        PanelPtr)win, "BadFilesButton");
488 1         win->MarkBadButton = (CButPtr)PANEL_GetNamedWgt((
                                          PanelPtr)win, "MarkBadButton");
489 1         win->AllowPartialButton = (CButPtr)PANEL_GetNamedWgt((
                                       PanelPtr)win, "AllowPartialButton");
490 1         win->CloseButton = (PButPtr)PANEL_GetNamedWgt((
                                          PanelPtr)win, "CloseButton");
491 1         win->StartButton = (PButPtr)PANEL_GetNamedWgt((
                                          PanelPtr)win, "StartButton");
492 1         win->HelpButton = (PButPtr)PANEL_GetNamedWgt((
                                          PanelPtr)win, "HelpButton");
493 1         WIN_SetWgtNfyHandler((WinPtr)win, (
                                 WgtPtr)win->BackupDateText, TED_NFYVALIDATE,
494 1               (
        WinWgtNfyHandlerProc)RestoreRestoreWin_ValidateBackupDateText);
495 1         WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->PrevBackupButton, TBUT_NFYHIT,
496 1               (
         WinWgtNfyHandlerProc)RestoreRestoreWin_HitPrevBackupButton);
497 1         WIN_SetWgtNfyHandler((WinPtr)win, (
                                     WgtPtr)win->CalendarButton, TBUT_NFYHIT,
498 1               (
           WinWgtNfyHandlerProc)RestoreRestoreWin_HitCalendarButton);
499 1         WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->NextBackupButton, TBUT_NFYHIT,
500 1               (
         WinWgtNfyHandlerProc)RestoreRestoreWin_HitNextBackupButton);
501 1         WIN_SetWgtNfyHandler((WinPtr)win, (
                                 WgtPtr)win->TemplateBox, CBOX_NFYELTSELECTED,
502 1               (
        WinWgtNfyHandlerProc)RestoreRestoreWin_EltSelectedTemplateBox);
503 1         WIN_SetWgtNfyHandler((WinPtr)win, (
                                   WgtPtr)win->PrimaryBox, CBOX_NFYELTSELECTED,
```

```
504 1                (
                WinWgtNfyHandlerProc)RestoreRestoreWin_EltSelectedPrimaryBox);
505 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->JunkTEd, TED_NFYVALIDATE,
506 1                (
                WinWgtNfyHandlerProc)RestoreRestoreWin_ValidateJunkTEd);
507 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                            WgtPtr)win->BackupListBox, LBOX_NFYCELLSTRING,
508 1                (
              WinWgtNfyHandlerProc)RestoreRestoreWin_CellStringBackupListBox);
509 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                             WgtPtr)win->BackupListBox, LBOX_NFYVALIDATE,
510 1                (
             WinWgtNfyHandlerProc)RestoreRestoreWin_ValidateBackupListBox);
511 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->PathTEd, TED_NFYVALIDATE,
512 1                (
                WinWgtNfyHandlerProc)RestoreRestoreWin_ValidatePathTEd);
513 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                                   WgtPtr)win->SearchButton, TBUT_NFYHIT,
514 1                (
                WinWgtNfyHandlerProc)RestoreRestoreWin_HitSearchButton);
515 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                                     WgtPtr)win->MarkButton, TBUT_NFYHIT,
516 1                (
                 WinWgtNfyHandlerProc)RestoreRestoreWin_HitMarkButton);
517 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                                   WgtPtr)win->UnmarkButton, TBUT_NFYHIT,
518 1                (
                WinWgtNfyHandlerProc)RestoreRestoreWin_HitUnmarkButton);
519 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                                 WgtPtr)win->MarkSummaryTab, TBUT_NFYHIT,
520 1                (
              WinWgtNfyHandlerProc)RestoreRestoreWin_HitMarkSummaryTab);
521 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                           WgtPtr)win->SelectedListBox, LBOX_NFYCELLSTRING,
522 1                (
           WinWgtNfyHandlerProc)RestoreRestoreWin_CellStringSelectedListBox);
523 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                            WgtPtr)win->SelectedListBox, LBOX_NFYVALIDATE,
524 1                (
            WinWgtNfyHandlerProc)RestoreRestoreWin_ValidateSelectedListBox);
525 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                          WgtPtr)win->RestoreItemsTarea, TED_NFYVALIDATE,
526 1                (
           WinWgtNfyHandlerProc)RestoreRestoreWin_ValidateRestoreItemsTarea);
527 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                           WgtPtr)win->RestoreSizeTarea, TED_NFYVALIDATE,
528 1                (
            WinWgtNfyHandlerProc)RestoreRestoreWin_ValidateRestoreSizeTarea);
529 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                               WgtPtr)win->BadFilesText, TED_NFYVALIDATE,
530 1                (
              WinWgtNfyHandlerProc)RestoreRestoreWin_ValidateBadFilesText);
531 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                                  WgtPtr)win->RemoveButton, TBUT_NFYHIT,
532 1                (
               WinWgtNfyHandlerProc)RestoreRestoreWin_HitRemoveButton);
533 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->ClearButton, TBUT_NFYHIT,
534 1                (
                WinWgtNfyHandlerProc)RestoreRestoreWin_HitClearButton);
535 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                                      WgtPtr)win->MediaTab, TBUT_NFYHIT,
536 1                (WinWgtNfyHandlerProc)RestoreRestoreWin_HitMediaTab);
537 1          WIN_SetWgtNfyHandler((WinPtr)win, (
```



```
                                 WgtPtr)win->MediaListBox, LBOX_NFYCELLSTRING,
538 1                (
           WinWgtNfyHandlerProc)RestoreRestoreWin_CellStringMediaListBox);
539 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->MediaListBox, LBOX_NFYVALIDATE,
540 1                (
             WinWgtNfyHandlerProc)RestoreRestoreWin_ValidateMediaListBox);
541 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                                   WgtPtr)win->ViewOptionsTab, TBUT_NFYHIT,
542 1                (
              WinWgtNfyHandlerProc)RestoreRestoreWin_HitViewOptionsTab);
543 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                                   WgtPtr)win->TypeSortButton, TBUT_NFYHIT,
544 1                (
              WinWgtNfyHandlerProc)RestoreRestoreWin_HitTypeSortButton);
545 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                                   WgtPtr)win->NameSortButton, TBUT_NFYHIT,
546 1                (
              WinWgtNfyHandlerProc)RestoreRestoreWin_HitNameSortButton);
547 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                                  WgtPtr)win->OwnerSortButton, TBUT_NFYHIT,
548 1                (
             WinWgtNfyHandlerProc)RestoreRestoreWin_HitOwnerSortButton);
549 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                                   WgtPtr)win->SizeSortButton, TBUT_NFYHIT,
550 1                (
              WinWgtNfyHandlerProc)RestoreRestoreWin_HitSizeSortButton);
551 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                                   WgtPtr)win->DateSortButton, TBUT_NFYHIT,
552 1                (
              WinWgtNfyHandlerProc)RestoreRestoreWin_HitDateSortButton);
553 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                                     WgtPtr)win->HiddenButton, TBUT_NFYHIT,
554 1                (
               WinWgtNfyHandlerProc)RestoreRestoreWin_HitHiddenButton);
555 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                                   WgtPtr)win->BadFilesButton, TBUT_NFYHIT,
556 1                (
              WinWgtNfyHandlerProc)RestoreRestoreWin_HitBadFilesButton);
557 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->MarkBadButton, TBUT_NFYHIT,
558 1                (
               WinWgtNfyHandlerProc)RestoreRestoreWin_HitMarkBadButton);
559 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                                 WgtPtr)win->AllowPartialButton, TBUT_NFYHIT,
560 1                (
           WinWgtNfyHandlerProc)RestoreRestoreWin_HitAllowPartialButton);
561 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                                     WgtPtr)win->CloseButton, TBUT_NFYHIT,
562 1                (
               WinWgtNfyHandlerProc)RestoreRestoreWin_HitCloseButton);
563 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                                     WgtPtr)win->StartButton, TBUT_NFYHIT,
564 1                (
              WinWgtNfyHandlerProc)RestoreRestoreWin_HitStartButton);
565 1          WIN_SetWgtNfyHandler((WinPtr)win, (
                                      WgtPtr)win->HelpButton, TBUT_NFYHIT,
566 1                (
               WinWgtNfyHandlerProc)RestoreRestoreWin_HitHelpButton);
567 1          /* )) CodeGen: WgtInitializations 808688
                                                                          */

569 1          /*
570 1           * Remove notifies for the TAB buttons, we have to leave them
571 1           * above so that ND regenerates OK.
572 1           */
```

```
575 1           WIN_RemoveWgtNfyHandler((WinPtr)win,
576 1                                   (WgtPtr)win->ViewOptionsTab,
577 1                                   TBUT_NFYHIT);
578 1           WIN_RemoveWgtNfyHandler((WinPtr)win,
579 1                                   (WgtPtr)win->MarkSummaryTab,
580 1                                   TBUT_NFYHIT);
581 1           WIN_RemoveWgtNfyHandler((WinPtr)win,
582 1                                   (WgtPtr)win->MediaTab,
583 1                                   TBUT_NFYHIT);
584 1           WIN_RemoveWgtNfyHandler((WinPtr)win,
585 1                                  (WgtPtr)win->PathTEd,
586 1                                  TED_NFYVALIDATE);

588 1           WIN_RemoveWgtNfyHandler((WinPtr)win,
589 1                                  (WgtPtr)win->TemplateBox,
590 1                                  CBOX_NFYELTSELECTED);
591 1           /*
592 1            * Reset the utility widgets to the utility style notifies
593 1            */

595 1           WGT_SetNfyProc ((WgtPtr)win, S_RestoreWinNfy);
596 1           WGT_SetNfyProc ((
                        WgtPtr)win->BackupDateText, GUTIL_OutputTED_DefNfy);
597 1           WGT_SetNfyProc((
                         WgtPtr)win->SelectedListBox, S_SelectedListBoxNfy);
598 1           WGT_SetNfyProc ((
                       WgtPtr)win->RestoreItemsTarea, GUTIL_OutputTED_DefNfy);
599 1           WGT_SetNfyProc ((
                        WgtPtr)win->RestoreSizeTarea, GUTIL_OutputTED_DefNfy);
600 1           WGT_SetNfyProc ((
                          WgtPtr)win->BadFilesText, GUTIL_OutputTED_DefNfy);
601 1           WGT_SetNfyProc((WgtPtr)win->MediaListBox, S_MediaListBoxNfy);
602 1           WGT_SetNfyProc((WgtPtr)win->PathTEd, S_PathTEdNfy);
603 1           WGT_SetNfyProc((WgtPtr)win->TemplateBox, S_TemplateBoxNfy);

605 1         /* Setup the path widget to use the utilities */
606 1         GTED_SetDefaults ((TEdPtr)win->PathTEd,
607 1                           TED_Alpha,
608 1                           GMAX_PATHNAME_LENGTH,
609 1                           0,
610 1                           0,
611 1                           STR_STD_FILENAME);

613 1           WIN_SetWgtNfyHandler((WinPtr)win, (
                              WgtPtr)win->HelpButton, TBUT_NFYGAINFOCUS,
614 1                    (
                    WinWgtNfyHandlerProc)RestoreRestoreWin_FocusHelpButton);
615       }
```



```
617       void      REST_RestoreWinLoadInit L0()
618 1     {
619 1               RestoreRestoreWinPtr win;

621 1               (void)RLIB_LoadLibFile("restore", "restore.dat");

623 1               win = (RestoreRestoreWinPtr)WIN_LoadSized(
                                                        "restore", "RestoreWin",
624 1                       sizeof(RestoreRestoreWinRec));
625 1               RestoreRestoreWin_Construct(win);

627 1               REST_RestoreWin = win;

629 1               WIN_Init((WinPtr)win);
630       }
```

```
632  static void REST_PrintUsageAndExit (Str command,
633                                      Str invalidOption)
634  1 {
635  1         /* Print out the bad option if one was given */
636  1   if (invalidOption != NULL)
637  2   {
638  2     STR_Printf ("%s: bad command line option \"%s\"\n\n",
639  2                 command,
640  2                 invalidOption);
641  1   }

643  1   /* Print out the useage */
644  1   STR_Printf (
                                          "usage:\n\t%s [-options ...
              ]\n\nwhere options include:\n\t-help\t\t\tprint out this
             message\n\t-display displayname\tX server to contact\n\n",
645  1               command);

647  1    fflush(stderr);
648  1    fflush(stdout);
649  1   _exit(0);
650     }
```



```
652  /* )) CodeGen: WindowSection RestoreWin                          */
653  /* (( CodeGen: WindowImplementationPlaceHolder ))                */

655  /* (( CodeGen: MainSection                                       */
656  /* ============================================================= = */
657  /* == Code for Main                                              */
658  /* ============================================================= = */

660  #include <nd.h>

662  ERR_DECLARE

664  /*
665   *  "main()" entrypoint
666   */

668  /*****************************************************************
669   * main
670   *
671   * Description:
672   *   This is the main routine, it will begin the restoral process.
673   *
674   * Parameters:
675   *   argc (I) - The count of the command line arguements.
676   *   argv (I) - The string list of the command line arguements.
677   *
678   * Returns:
679   *   None.
680   *
681   *****************************************************************/

683  int      main L2(int, argc, char**, argv)
684  1 {
685  1   int       i;                        /* Loop counter */
686  1   BoolEnum  traceOn = BOOL_FALSE;
                                  /* Flag if tracing should be turned on */
687  1   char     *displayString;
                                      /* String for DISPLAY env variable */
688  1   int       nkey = 0;                 /* Number of keys */
689  1   int       key1;                     /* First Key */
690  1   int       key2;                     /* Second Key */
691  1   int       inputQ;                   /* Shared help queue for input */
692  1   int       outputQ;
                                           /* Shared help queue for output */
693  1   BoolEnum  usingIPC = BOOL_FALSE;
                                 /* Flag if we are talking with main view */
694  1   BoolEnum  sharedHelp = BOOL_FALSE;
                                  /* Flag if sharing help with main view */
695  1   BoolEnum  synchronize = BOOL_FALSE;
                                  /* Flag if we shoul run in X sync mode */
696  1   char     *message;
                                       /* Message to send to main view */
697  1   IPCLHandle handle;
                                       /* Handle to comms to main view */
698  1   ipclStatus status;
                                  /* pass/fail status of ipc connection */
699  1   BoolEnum   setColors = BOOL_FALSE;
                                  /* Flag if user specified color scheme */
700  1   Str        colorStr;
```

```
                                        /* Color schema string from args */
  Int         colorRet = RESOURCE_NULL_ARGC;
                                        /* Return val from color func */
  GALERT_WinHandle synchHandle = NULL;

  /* Register this program */
  GUTIL_RegisterProgram (argv[0]);

  /* Set the restore from client flag to false for starters */
  REST_RestoreFromClient = BOOL_FALSE;

  /* Initialize the global color value for color schemes */
  colorVal = argc + 1 ;
  colorArgs = argc;

  /* Initialize the variable REST_RestoreClient */
  STR_Cpy (REST_RestoreClient, "");

  /* Loop through the command line arguments */
  for (i=1; i<argc; i++)
  {
    /* Check if this is a HELP option */
    if (strcmp(argv[i], "-"HELP_OPTION) == 0)
    {
      REST_PrintUsageAndExit (basename(argv[0]), NULL);
    }

    /* Check if this is a version option */
    else if (strcmp(argv[i], "-"VERSION_OPTION) == 0)
    {
      GUTIL_PrintVersionAndExit ();
    }

    /* Check if this is the DISPLAY option */
    else if (strcmp(argv[i], "-" DISPLAY_OPTION) == 0)
    {
      /* Get the next argument and set the display variable to it */
      i++;
      GUTIL_SetDisplay (argv[i]);
    }

    /* Check if this is the Client side restoral option */
    else if (strcmp(argv[i], "-" FROM_CLIENT_OPTION) == 0)
    {

      /* Flag that this is client side restoral */
      REST_RestoreFromClient = BOOL_TRUE;

      /* Get the client name */
      i++;
      STR_Cpy (REST_RestoreClient, argv[i]);
    }

    /* Check for the -color option */
/*
 * COLOR SCHEME NOT SUPPORTED IN THIS RELEASE!!!!
 *

    else if (strncmp(argv[i],
                     "-" RES_COLOR_OPTION,
                     strlen(RES_COLOR_OPTION) + 1) == 0)
    {
      i++;
      setColors = BOOL_TRUE;
      colorStr = (Str) argv[i];
    }
```



```
 *
 */

    /* If this is the IPC Key option, get the keys */
    else if (strcmp(argv[i], "-" IPC1_KEY_OPTION) == 0)
    {
      nkey = atoi(argv[++i]);
      key1 = atoi(argv[++i]);
      if(nkey == 2)
        key2 = atoi(argv[++i]);
      else
        key2 = key1;
      usingIPC = BOOL_TRUE;
    }

    /* If this is the Help Queue option, get the queues */
    else if (strcmp(argv[i], "-" RESTORE_HELP_Q_OPT) == 0)
    {
      inputQ = atoi(argv[++i]);
      outputQ = atoi(argv[++i]);
      sharedHelp = BOOL_TRUE;
    }

    /* If this is the trace option */
    else if (strncmp(argv[i], "-" TRACE_OPTION, strlen(
                                                TRACE_OPTION)+1) == 0)

    {
      /* Turn on tracing */
      traceOn = BOOL_TRUE;
    }

    /* Check if this is the client list option */
    else if (strcmp(argv[i], "-" RESTORE_CLIENT_OPT) == 0)
    {
      /* skip all following arguments up to the next option */
      i++;
      while ((i<argc) && (argv[i][0] != '-'))
      {
        i++;
      }

      /* If we exitted the loop because of a new option,
                                                rewind to the option */

      if (i<argc)
        i--;
    }

    /* Check if this is the work item list option */
    else if (strcmp(argv[i], "-" RESTORE_WI_OPT) == 0)
    {
      /* skip all following arguments up to the next option */
      i++;
      while ((i<argc) && (argv[i][0] != '-'))
      {
        i++;
      }

      /* If we exitted the loop because of a new option,
                                                rewind to the option */

      if (i<argc)
        i--;
    }

    /* Check if this is the synchronize option */
    else if (strncmp(argv[i], "-"SYNC_OPTION, strlen(
                                                SYNC_OPTION) + 1) == 0)
```

```
827  3      {
828  3        synchronize = BOOL_TRUE;
829  2      }

831  2      else
832  3      {
833  3        REST_PrintUsageAndExit (basename(argv[0]), argv[i]);
834  2      }
835  1    }

837  1    /* Check the initial Environment */
838  1    GUTIL_CheckInitEnv (argv[0]);

840  1    /* default initialization */
841  1    ERR_MAININIT;
842  1    ERR_MODULEUSE;

844  1    /*
845  1     * Initialize ND stuff
846  1     */

848  1    ND_Init (argc, argv);

850  1    /* Initialize the utilities */
851  1    GUTIL_Initialize ();

853  1    /* Install the generic signal handlers */
854  1    GUTIL_AddGenericSignalHandlers ((
                            GUTIL_SignalHandlerProc) REST_SignalHandler);

856  1    /* Initialize the use of RPCs in this process */
857  1    GUTIL_InitializeRPC ();

859  1    /* if there exists an argument after the -color */
860  1    if (setColors)
861  1      colorRet = GUTIL_ReadResFile (colorStr, BOOL_TRUE);

863  1    /* if there wasn't an argument or if the read failed */
864  1    if (colorRet != RESOURCE_FILE_OK)
865  1      GUTIL_ReadResFile ("", BOOL_TRUE);

867  1    /* Set the defaults to cover people who don't set correctly */
868  1    GUTIL_SetResourcesToDefaults (BOOL_TRUE);

870  1    /* Set the running directory */
871  1    GUTIL_SetRunningDirectory (basename (argv[0]));

873  1    /* Never, I mean NEVER, let the user see OpenLook (yuk!) */
874  1    if (DSPLY_GetLook () == DSPLY_LOOKOPENLOOK)
875  1      DSPLY_SetLook (DSPLY_LOOKMOTIF);

877  1    /* Initialize the restore components */
878  1    REST_Initialize ();

880  1    /* Show the about box while initializing if not from edm */
881  1    if (!usingIPC)
882  2    {
883  2      ABOUT_DisplayBanner (NULL);
884  2      EVENT_Update ();
885  1    }
886  1    /* Otherwise, show an initializing message */
887  1    else
888  2    {
889  2      synchHandle = GALERT_DisplaySynchronousWait (NULL,
890  2                                    REST_GetErrorString (
                                                    REST_INIT_TITLE),
```



```
891  2                                    GICON_GetIcon (I_WAIT),
892  2                                    REST_GetErrorString (
                                                    REST_INIT_MESSAGE),
893  2                                    BOOL_FALSE);
894  2      EVENT_Update ();
895  1    }

897  1    /*
898  1     * Determine the list of clients to display in the file manager
899  1     */

901  1    REST_StartClientList ();
902  1    for (i=1; i<argc; i++)
903  2    {
904  2      /* Check if this is an option */
905  2      if (argv[i][0] == '-')
906  3      {
907  3        /* If this is the client list option */
908  3        if (strcmp(argv[i], "-" RESTORE_CLIENT_OPT) == 0)
909  4        {

911  4          /* Get all following arguments up to the next option */
912  4          i++;
913  4          while ((i<argc) && (argv[i][0] != '-'))
914  5          {
915  5            /* This is another client to add to the list */
916  5            REST_AddClient (argv[i]);
917  5            i++;
918  4          }

920  4          /* OK, we can back up one */
921  4          i--;
922  3        }

924  2      }
925  1    }

927  1    /*
928  1     * Create the list of selected work items
929  1     */

931  1    REST_StartWiList ();
932  1    for (i=1; i<argc; i++)
933  2    {
934  2      /* Check if this is an option */
935  2      if (argv[i][0] == '-')
936  3      {
937  3        /* If this is the client list option */
938  3        if (strcmp(argv[i], "-" RESTORE_WI_OPT) == 0)
939  4        {

941  4          /* Get all following arguments up to the next option */
942  4          i++;
943  4          while ((i<argc) && (argv[i][0] != '-'))
944  5          {
945  5            /* This is another work item to add to the list */
946  5            REST_AddWi (argv[i]);
947  5            i++;
948  4          }

950  4          /* OK, we can back up one */
951  4          i--;
952  3        }

954  2      }
955  1    }
```

```
957 1   /* If syncronous mode, set mode */
958 1   if (synchronize)
959 2   {
960 2     XSynchronize (X_Display(), BOOL_TRUE);
961 1   }

963 1   /* IF tracing is on */
964 1   if (traceOn)
965 2   {
966 2     /* start with everything */
967 2     TRACESETFLAGS(V_TRACE_EVERYTHING)

969 2     /* start tracing */
970 2     TRACESTART
971 2     TRACESTARTFILEIO

973 2     /* Display the trace controls window */
974 2     TRACEPOPUPCONTROLS
975 1   }

977 1   /* If we are talking with mainview, tell it we're up */
978 1   if (usingIPC)
979 2   {
980 2     status = ipclOpen (&handle, key1, key2);
981 2     if ((handle != NULL) && (status != IPCL_FAILURE))
982 3     {
983 3       message = (char *) GUTIL_Malloc (strlen(IPC_CONNECT_STRING) + 1);
984 3       STR_Cpy (message, IPC_CONNECT_STRING);
985 3       ipclSendMessage (handle,
986 3                        IPCL_NOWAIT,
987 3                        1,
988 3                        message,
989 3                        strlen(IPC_CONNECT_STRING) + 1);
990 3       GUTIL_Free (message);
991 2     }
992 1   }

994 1   /* Display the restore window */
995 1   REST_Display ();

997 1   /* If we are sharing help with mainview,
                                    talk to the already running help */
998 1   if (sharedHelp)
999 2   {
1000 2    EDMHELP_InitQueues (inputQ, outputQ);
1001 1  }

1003 1  /* Remove the about box */
1004 1  if (!usingIPC)
1005 2  {
1006 2    ABOUT_TerminateWin ();
1007 2    EVENT_Update ();
1008 1  }
1009 1  else if (synchHandle != NULL)
1010 2  {
1011 2    GALERT_CancelSynchDialog (synchHandle);
1012 2    EVENT_Update ();
1013 1  }

1015 1  /* Put up the window */
1016 1  WIN_Show((WinPtr)REST_RestoreWin);

1018 1  /* Begin the event processing */
1019 1  ND_Run();
```



```
1021 1  /*
1022 1   * OK, we're outta here!  Clean up and go home
1023 1   */

1025 1  /* Close help */
1026 1  EDMHELP_End ();

1028 1  /* default termination */
1029 1  ND_Exit ();

1031 1  /* Exit the process */
1032 1  return EXIT_OK;
1033    }
```

```
1034  /* )) CodeGen: MainSection                                    */
1035  /* (( CodeGen: MainPlaceHolder ))                             */
```

```
/*****************************************************************
 * restMgr.c
 *
 *
 * Copyright 1996 by Epoch Systems, Inc.
 *
 *
 * Mission Statement:
 *   This file contains the functions necessary for the EDM Restore
                                                              window.
 *
 * Required includes:
 *   None
 *
 * Compile-Time Options:
 *   N/A
 *
 *
 * RCS Information:
 *     $RCSfile$
 *     $Revision$
 *     $Date$
 *****************************************************************/

#define  ERR_LIB  RESTORE

#include <esl/c_portable.h>

#include <stdlib.h>

#include <libgen.h>
#include <time.h>

#include <apppub.h>
#include <eventpub.h>
#include <rlibpub.h>
#include <gwpub.h>
#include <respub.h>
#include <lboxpub.h>
#include <mbarpub.h>
#include <panelpub.h>
#include <tareapub.h>
#include <tbutpub.h>
#include <tedpub.h>
#include <winpub.h>
#include <strlpub.h>
#include <drawpub.h>
#include <dsplypub.h>
#include <ibutpub.h>
#include <arraypub.h>

#include "eerrno/e_errno.h"
#include "util/esl_string.h"
#include <restore/restore_api.h>
#define REST_MGR_INIT
#include "restore/restMgr.h"
#undef REST_MGR_INIT
#include "restore.h"
#include "restoreP.h"
#include "restCalendar.h"
#include "restSearch.h"
#include "restSelMgr.h"
#include "restFileMgr.h"
#include "restUtils.h"
#include "restCBMgr.h"
#include "restAPIUtils.h"
```



```
#include "gutil/timeUtils.h"
#include "gutil/iconDefs.h"
#include "gutil/iconUtils.h"
#include "gutil/winUtils.h"
#include "gutil/resUtils.h"
#include "gutil/miscUtils.h"
#include "gutil/fileMgr.h"
#include "gutil/tabDefs.h"
#include "gutil/guiDefines.h"
#include "gutil/guiUtils.h"
#include "gutil/hostUtils.h"
#include "gutil/cboxUtils.h"
#include "gutil/codeTracer.h"
#include "gutil/alertMgr.h"
#include "help/helpDefs.h"
#include "help/helpIpc.h"
#include "ipc/ipcl.h"

ERR_EXTERN
ERR_INMODULE("restore")

/*************
 * Constants *
 *************/

#define MAX_TIMES_BUFFER       64
#define REST_MARK_THRESHOLD  1000
#define CONNECT_TIMEOUT_SEC    60

/************************
 * Local Data Structures *
 ************************/

/************************
 * Local Global Variables *
 ************************/

/* Current list of clients */
static RestoreClientPtr currentClientList = NULL;

/* Current list of work items */
static RestoreWorkItemPtr currentWIList = NULL;

/* Flags for current state of processing */
static BoolEnum  updatingDate = BOOL_FALSE;

/* Static variables for mark progress */
static GALERT_WinHandle synchMarkHandle = NULL;

/* Handle to the current fill in progress dialog */
static GALERT_WinHandle synchFillHandle = NULL;

/* Forward declaration of signal handler */
void REST_SignalHandler (int sig);

/* Forward declaration of validate work item routine */
static BoolEnum REST_ValidateWorkItem (GREST_Object  workItemObject,
                                       eerrno_ty    *errorCode);

static BoolEnum GREST_IsObjectMarked (serverHandle serverHandle,
                                      GREST_Object object)
{
  unsigned long numChecked;
  boolean_ty    isMarked = BOOL_FALSE;

  if ((serverHandle != NULL) && (object != NULL))
```

```
132 2   {
133 2     EDMRST_IsObjectMarked (serverHandle,
134 2                            1,
135 2                            &object,
136 2                            &numChecked,
137 2                            &isMarked);
138 1   }

140 1   return (isMarked);
141   }
```



```
143   /*********************************************************************
144    * REST_DisplayBackupDate
145    *
146    * Description:
147    *   This routine will display the current backup date and set the
148    *   date buttons to their correct states.
149    *
150    * Parameters:
151    *   None.
152    *
153    * Returns:
154    *   None.
155    *
156    *********************************************************************/

158   void  REST_DisplayBackupDate (void)
159 1 {
160 1   Char      dateString[GMAX_OBJECT_LENGTH]; /* Date to display */
161 1   time_t    thisTime;                       /* Current backup time */
162 1   eerrno_ty eerrno;                         /* Error status */

164 1   /* Get the current backup time */
165 1   if ((eerrno = EDMRST_GetCurrentBackupTime(
                                    GREST_Handle, &thisTime)) == E_SUCCESS)
166 2   {

168 2     /* Print the time and date to the date string */
169 2     STR_Sprintf (dateString, "%s", REST_GetTimeDateString (thisTime));

171 1   }
172 1   else
173 2   {
174 2     /* Couldn't get a backup time, must not have a work item yet */
175 2     STR_Sprintf (dateString, "");
176 2     thisTime = 0;
177 1   }

179 1   /* Set the date text label */
180 1   TED_SetStr ((TEdPtr)REST_RestoreWin->BackupDateText, dateString);

182 1   /* Update the sensitivity of the date buttons */
183 1   REST_UpdateDateButtons ();
184   }
```

```
186  /*****************************************************************
187   * REST_UpdateBackupDate
188   *
189   * Description:
190   *   This routine will update the current work item and the displayed
191   *   date after a date change.
192   *
193   * Parameters:
194   *   None.
195   *
196   * Returns:
197   *   None.
198   *
199   *****************************************************************/

201  void  REST_UpdateBackupDate (void)
202 1{

204 1   /* Flag that we are updating the backup date */
205 1   updatingDate = BOOL_TRUE;

207 1   /* Re-Read the current work item */
208 1   REST_ReReadWorkItem (currentWorkItemInfo);

210 1   /* Display the new backup date */
211 1   REST_DisplayBackupDate ();

213 1   /* Flag that we are no longer updating the backup date */
214 1   updatingDate = BOOL_FALSE;
215  }
```



```
217  /*****************************************************************
218   * REST_UpdateBackupTemplates
219   *
220   * Description:
221   *   This routine will update the template and trailset boxes
222   *   to the current choices available.
223   *
224   * Parameters:
225   *   info (I) - Work-Item info record to get the templates for
226   *
227   * Returns:
228   *   None.
229   *
230   *****************************************************************/

232  void  REST_UpdateBackupTemplates (RestoreInfoPtr info)
233 1{
234 1   template_name_ty    currentTemplate;
                                                  /* Current template */
235 1   template_name_ty    templates[TEMPLATE_BUFFER_LENGTH];
                                                    /* New templates */
236 1   BoolEnum            isAlternate;
                                                  /* Alternate flag */
237 1   BoolEnum            exists = BOOL_FALSE;
                                                     /* Exists flag */
238 1   BoolEnum            currentFound = BOOL_FALSE;
                                                     /* Found flag */
239 1   long                cookie = INIT_COOKIE;
                                                  /* The magic cookie */
240 1   short               numEntries;
                                                     /* Number found */
241 1   Int                 i;
                                                     /* Loop counter */
242 1   eerrno_ty           eerrno;
                                                     /* Error status */


244 1   /* Validate the object passed in */
245 1   if ((info != NULL) &&
246 1       (info->type == REST_WorkItem) &&
247 1       (info->restoreObject != NULL))
248 2   {

250 2     /* First, clear out any old Templates: */
251 2     CBOX_GoFirst(REST_RestoreWin->TemplateBox);
252 2     while (CBOX_IsOk (REST_RestoreWin->TemplateBox))
253 3     {
254 3       CBOX_GoFirst (REST_RestoreWin->TemplateBox);
255 3       CBOX_CurRemoveElt (REST_RestoreWin->TemplateBox);
256 2     }

258 2     /* get the current template */
259 2     if (EDMRST_GetCurrentTemplate (GREST_Handle,
260 2                                    currentTemplate,
261 2                                    &isAlternate) != E_SUCCESS)
262 3     {
263 3       STR_Cpy (currentTemplate, "");
264 3       isAlternate = BOOL_FALSE;
265 2     }

267 2     /* Get all the templates for the new work item */
268 2     while (cookie != DONE_COOKIE)
269 3     {
270 3       numEntries = 0;
271 3       if ((eerrno = EDMRST_GetTopLevelTemplates (GREST_Handle,
272 3                                                  info->restoreObject,
```

```
273 3                                                   TEMPLATE_BUFFER_LENGTH,
274 3                                                     templates,
275 3                                                     &numEntries,
276 3                                                     &cookie)) ==
                                                                  E_SUCCESS)
277 4       {

279 4         /* Add each entry to the template box */
280 4         CBOX_GoFirst(REST_RestoreWin->TemplateBox);
281 4         for (i=0; i<numEntries; i++)
282 5         {
283 5           CBOX_CurAddElt (REST_RestoreWin->TemplateBox, (
                                                              ClientPtr)NULL);
284 5           CBOX_CurSetLabel (
                              REST_RestoreWin->TemplateBox, templates[i]);

286 5           /* If this is the current template, select it */
287 5           if (STR_Cmp (templates[i], currentTemplate) == CMP_EQUAL)
288 6           {
289 6             REST_SelectCurrentTemplate ();
290 6             currentFound = BOOL_TRUE;
291 5           }
292 5           CBOX_GoNext (REST_RestoreWin->TemplateBox);
293 4         }

295 4         if ((!currentFound) && (STR_Cmp (
                                       currentTemplate, "") != CMP_EQUAL))
296 5         {
297 5           CBOX_CurAddElt (REST_RestoreWin->TemplateBox, (
                                                              ClientPtr)NULL);
298 5           CBOX_CurSetLabel (
                           REST_RestoreWin->TemplateBox, currentTemplate);
299 5           REST_SelectCurrentTemplate ();
300 5           numEntries++;
301 5           currentFound = BOOL_TRUE;
302 4         }

304 4         /* If we didn't find it (
                                  probably no savesets) select the first */
305 4         if ((!currentFound) && (numEntries > 0))
306 5         {
307 5           CBOX_GoFirst (REST_RestoreWin->TemplateBox);
308 5           REST_SelectCurrentTemplate ();
309 4         }
310 3       }
311 3       else
312 4       {
313 4         /* At least add the current template */
314 4         if (STR_Cmp (currentTemplate, "") != CMP_EQUAL)
315 5         {
316 5           CBOX_CurAddElt (REST_RestoreWin->TemplateBox, (
                                                              ClientPtr)NULL);
317 5           CBOX_CurSetLabel (
                           REST_RestoreWin->TemplateBox, currentTemplate);
318 5           REST_SelectCurrentTemplate ();
319 4         }

321 4         /* Just get out */
322 4         cookie = DONE_COOKIE;
323 3       }
324 2     }

326 2     /* First, clear out any old trails: */
327 2     CBOX_GoFirst(REST_RestoreWin->PrimaryBox);
328 2     while (CBOX_IsOk (REST_RestoreWin->PrimaryBox))
```



```
329 3     {
330 3       CBOX_GoFirst (REST_RestoreWin->PrimaryBox);
331 3       CBOX_CurRemoveElt (REST_RestoreWin->PrimaryBox);
332 2     }

334 2     /* Add the primary trail (always exists) */
335 2     CBOX_GoFirst(REST_RestoreWin->PrimaryBox);
336 2     CBOX_CurAddElt (REST_RestoreWin->PrimaryBox, (ClientPtr)NULL);
337 2     CBOX_CurSetLabel (REST_RestoreWin->PrimaryBox,
338 2                       (Str)STRL_GetNthStr(REST_TrailNameList,
339 2                                           REST_PRIMARY_TRAIL_INDEX));
340 2     CBOX_CurSetId (REST_RestoreWin->PrimaryBox, 1);

342 2     /* Add the alternate trail if it exists */
343 2     EDMRST_DoesAlternateExist (GREST_Handle,
344 2                                info->restoreObject,
345 2                                currentTemplate,
346 2                                &exists);
347 2     if (exists)
348 3     {
349 3       CBOX_GoNext (REST_RestoreWin->PrimaryBox);
350 3       CBOX_CurAddElt (REST_RestoreWin->PrimaryBox, (ClientPtr)NULL);
351 3       CBOX_CurSetLabel (REST_RestoreWin->PrimaryBox,
352 3                         (Str)STRL_GetNthStr(REST_TrailNameList,
353 3                                             REST_ALTERNATE_TRAIL_INDEX));
354 3       CBOX_CurSetId (REST_RestoreWin->PrimaryBox, 2);
355 2     }

357 2     /* Show whether or not it is the alternate */
358 2     CBOX_GoFirst (REST_RestoreWin->PrimaryBox);
359 2     if (isAlternate)
360 3     {
361 3       CBOX_GoNext (REST_RestoreWin->PrimaryBox);
362 2     }
363 2     REST_SelectCurrentTrail ();

365 1   }

367   }
```

```
369  /*****************************************************************
370   * REST_UpdateChildMarks
371   *
372   * Description:
373   *   This routine will update the marked flag for all children and
374   *   recursively for their children for the given object.
375   *
376   * Parameters:
377   *   parentObject (I) - The object whose children need to be updated
378   *
379   * Returns:
380   *   None.
381   *
382   *****************************************************************/

384  void REST_UpdateChildMarks (RestoreInfoPtr parentObject)
385 1 {
386 1   RestoreInfoPtr tmpInfo;   /* Info to walk the list with */
387 1   unsigned long  numChecked;

389 1   /* Validate the input */
390 1   if (parentObject != NULL)
391 2   {
392 2     /* Walk the children list */
393 2     tmpInfo = parentObject->children;
394 2     while (tmpInfo != NULL)
395 3     {

397 3       /* Determine if this object is marked */
398 3       if (tmpInfo->restoreObject != NULL)
399 4       {
400 4         tmpInfo->marked = GREST_IsObjectMarked (GREST_Handle,
401 4                                                 tmpInfo->restoreObject);
402 3       }

404 3       /* Update the marks for the children of this object */
405 3       REST_UpdateChildMarks (tmpInfo);

407 3       /* Go to the next child */
408 3       tmpInfo = tmpInfo->next;
409 2     }
410 1   }
411   }
```



```
413  /*****************************************************************
414   * REST_UpdateObjectMarks
415   *
416   * Description:
417   *   This routine will update the marked flag the given object and
418   *   all its children.
419   *
420   * Parameters:
421   *   parentObject (
                I) - The top level object whose children need to be updated
422   *
423   * Returns:
424   *   None.
425   *
426   *****************************************************************/

428  void REST_UpdateObjectMarks (RestoreInfoPtr parentObject)
429 1 {
430 1   RestoreInfoPtr nextChild;
                                          /* Info to walk the list with */
431 1   BoolEnum       allMarked = BOOL_TRUE;
                                    /* Flag if all children are marked */

433 1   /* Validate the input */
434 1   if (parentObject != NULL)
435 2   {

437 2     /* Determine if this object is marked */
438 2     if (parentObject->restoreObject != NULL)
439 3     {
440 3       parentObject->marked = GREST_IsObjectMarked (GREST_Handle,
441 3                                                    parentObject->restoreObject);
442 2     }

444 2     /* Update the marks for the children of this object */
445 2     REST_UpdateChildMarks (parentObject);
446 1   }

448 1   /* See if the entire workitem is marked */
449 1   if (currentWorkItemInfo != NULL)
450 2   {
451 2     nextChild = currentWorkItemInfo->children;
452 2     while (allMarked && (nextChild != NULL))
453 3     {
454 3       if (!nextChild->marked)
455 4       {
456 4         allMarked = BOOL_FALSE;
457 3       }
458 3       else
459 4       {
460 4         nextChild = nextChild->next;
461 3       }
462 2     }
463 2     currentWorkItemInfo->marked = allMarked;
464 1   }
465   }
```

```
467  /*****************************************************************
468   * REST_ClearChildMarks
469   *
470   * Description:
471   *   This routine will clear the marked flag for all children and
472   *   recursively for their children for the given object.
473   *
474   * Parameters:
475   *   parentObject (I) - The object whose children need to be cleared
476   *
477   * Returns:
478   *   None.
479   *
480   *****************************************************************/

482  void REST_ClearChildMarks (RestoreInfoPtr parentObject)
483 1{
484 1  RestoreInfoPtr tmpInfo;  /* Info to walk the list with */

486 1  /* Validate the input */
487 1  if (parentObject != NULL)
488 2  {

490 2    /* Walk the children list */
491 2    tmpInfo = parentObject->children;
492 2    while (tmpInfo != NULL)
493 3    {

495 3      /* Clear the marked flag for this object */
496 3      if (tmpInfo->marked)
497 4      {
498 4        tmpInfo->marked = BOOL_FALSE;
499 4        GFMGR_UpdateObject (REST_GetFMgrContext(), (
                                                GFMGR_Object)tmpInfo);
500 3      }

502 3      /* If this object is in the selected list, remove it */
503 3      if (REST_IsItemSelected (tmpInfo) &&
504 3          (tmpInfo->restoreObject != NULL))
505 4      {
506 4        REST_DeselectInfo (tmpInfo->restoreObject, 0);
507 3      }

509 3      /* Clear the marks for the children of this object */
510 3      REST_ClearChildMarks (tmpInfo);

512 3      /* Go to the next child */
513 3      tmpInfo = tmpInfo->next;
514 2    }
515 1  }
516  }
```



```
518  /*****************************************************************
519   * REST_ClearObjectMarks
520   *
521   * Description:
522   *   This routine will clear the marked flag for the given object
523   *   and for all descendants of the given object.
524   *
525   * Parameters:
526   *   parentObject (I) - The object whose children need to be cleared
527   *
528   * Returns:
529   *   None.
530   *
531   *****************************************************************/

533  void REST_ClearObjectMarks (RestoreInfoPtr parentObject)
534 1{

536 1  /* Validate the input */
537 1  if (parentObject != NULL)
538 2  {

540 2    /* Clear the marked flag for this object */
541 2    if (parentObject->marked)
542 3    {
543 3      parentObject->marked = BOOL_FALSE;
544 3      GFMGR_UpdateObject (REST_GetFMgrContext(), (
                                                GFMGR_Object)parentObject);
545 2    }

547 2    /* If this object is in the selected list, remove it */
548 2    if (REST_IsItemSelected (parentObject) &&
549 2        (parentObject->restoreObject != NULL))
550 3    {
551 3        REST_DeselectInfo (parentObject, 0);
552 2    }

554 2    /* Clear the marks for the children of this object */
555 2    REST_ClearChildMarks (parentObject);

557 1  }

559 1  /* Clear out any selection data */
560 1  REST_ClearMarkedInfo ();

562  }
```

```
564   /*****************************************************************
565    * REST_GetCurrentClientInfo
566    *
567    * Description:
568    *   This routine will return the current client object.
569    *
570    * Parameters:
571    *   None.
572    *
573    * Returns:
574    *   The current cliennt object, or NULL if none.
575    *
576    *****************************************************************/

578   RestoreInfoPtr REST_GetCurrentClientInfo (void)
579 1 {
580 1   RestoreInfoPtr returnInfo;  /* Info to return */

582 1   /* If we are currently working with a work item,
                                                  return it's parent */
583 1   if (currentWorkItemInfo != NULL)
584 1     returnInfo = currentWorkItemInfo->parent;

586 1   /* Else, there is no current client */
587 1   else
588 1     returnInfo = NULL;


591 1   /* Return the determined info */
592 1   return (returnInfo);
593   }
```



```
595   /*****************************************************************
596    * REST_GetCurrentWorkItem
597    *
598    * Description:
599    *   This routine will return the current work-item restorable object.
600    *
601    * Parameters:
602    *   None.
603    *
604    * Returns:
605    *   The current work item object, or NULL if none.
606    *
607    *****************************************************************/

609   GREST_Object REST_GetCurrentWorkItem (void)
610 1 {
611 1   GREST_Object returnObject;  /* The work item object to return */

613 1   /* If we are currently looking at a work-item */
614 1   if (currentWorkItemInfo != NULL)
615 1     /* return the current work item object */
616 1     returnObject = currentWorkItemInfo->restoreObject;
617 1   else
618 1     /* return NULL */
619 1     returnObject = NULL;

621 1   return (returnObject);
622   }
```

```
624   /*****************************************************************
625    * REST_CreateClientInfo
626    *
627    * Description:
628    *   This routine will create the data for the given client.
629    *
630    * Parameters:
631    *   clientName (I) - The name of the client
632    *
633    * Returns:
634    *   The allocated data for the given client.
635    *
636    *****************************************************************/

638   RestoreInfoPtr  REST_CreateClientInfo (Str clientName)
639 1 {
640 1   RestoreInfoPtr     newInfo;
                                        /* Info created for client */
641 1   BuCfgPlatformType  platformType = BUCFG_PLATFORM_UNKNOWN;
                                                  /* Platform type */


643 1   /* Create a new info record */
644 1   newInfo = (RestoreInfoPtr) GUTIL_Malloc (sizeof(RestoreInfoRec));
645 1   newInfo->parent = NULL;
646 1   newInfo->children = NULL;
647 1   newInfo->next = NULL;

649 1   /* Copy in the fields */
650 1   newInfo->name = esl_strdup (clientName);
651 1   newInfo->type = REST_Client;

653 1   /*
654 1    * Determine the platform type and get the appropriate icon
655 1    */

657 1   if (EDMRST_GetHostPlatformType (
                  GREST_Handle, clientName, &platformType) == E_SUCCESS)
658 2   {
659 2     switch (platformType)
660 3     {
661 3       case BUCFG_PLATFORM_UNIX:
662 3         newInfo->icon = GICON_GetIconBySize (I_UNIXCLIENT, ICON_SMALL);
663 3         break;
664 3       case BUCFG_PLATFORM_OS2:
665 3         newInfo->icon = GICON_GetIconBySize (I_OS2CLIENT, ICON_SMALL);
666 3         break;
667 3       case BUCFG_PLATFORM_NETWARE:
668 3         newInfo->icon = GICON_GetIconBySize (
                                          I_NETWARECLIENT, ICON_SMALL);
669 3         break;
670 3       case BUCFG_PLATFORM_WNT:
671 3         newInfo->icon = GICON_GetIconBySize (
                                            I_WINNTCLIENT, ICON_SMALL);
672 3         break;
673 3       case BUCFG_PLATFORM_VMS:
674 3         newInfo->icon = GICON_GetIconBySize (I_VMSCLIENT, ICON_SMALL);
675 3         break;
676 3       default:
677 3         newInfo->icon = GICON_GetIconBySize (
                                          I_UNKNOWNCLIENT, ICON_SMALL);
678 2     }
679 1   }
680 1   else
681 2   {
682 2     newInfo->icon = GICON_GetIconBySize (I_UNKNOWNCLIENT, ICON_SMALL);
```



```
683 1   }

685 1   newInfo->opened = BOOL_FALSE;
686 1   newInfo->restoreObject = NULL;
687 1   newInfo->status = Backup_Good;
688 1   newInfo->backupTime = 0;
689 1   newInfo->errorString = NULL;

691 1   return (newInfo);
692   }
```

```
694   /*****************************************************************
695    * REST_CreateErrorInfo
696    *
697    * Description:
698    *   This routine will create the data for an error object
699    *
700    * Parameters:
701    *   errString (I) - The error string for the new object
702    *   parent    (I) - The parent of the error object
703    *
704    * Returns:
705    *   The allocated data for the error object
706    *
707    *****************************************************************/

709   RestoreInfoPtr  REST_CreateErrorInfo (Str            errString,
710                                         RestoreInfoPtr parent)
711 1 {
712 1   RestoreInfoPtr     newInfo;
                           /* New info created for the error Object */


714 1   /* Create the new object */
715 1   newInfo = (RestoreInfoPtr) GUTIL_Malloc (sizeof(RestoreInfoRec));
716 1   newInfo->parent = parent;
717 1   newInfo->children = NULL;
718 1   newInfo->next = NULL;

720 1   /* Fill in the data fields */
721 1   if (errString != NULL)
722 1     newInfo->name = esl_strdup (errString);
723 1   else
724 1     newInfo->name = NULL;
725 1   newInfo->type = REST_ErrorObject;
726 1   newInfo->opened = BOOL_FALSE;
727 1   newInfo->marked = BOOL_FALSE;
728 1   newInfo->restoreObject = NULL;
729 1   newInfo->status = Backup_Good;
730 1   newInfo->backupTime = 0;
731 1   newInfo->errorString = NULL;

733 1   newInfo->icon = REST_FailedIcon;

735 1   /* Return the new object */
736 1   return (newInfo);
737   }
```



```
739   /*****************************************************************
740    * REST_CreateWorkItemInfo
741    *
742    * Description:
743    *   This routine will create the data for the give work item object.
744    *
745    * Parameters:
746    *   object (I) - The work item object.
747    *   parent (I) - The parent of the new object
748    *
749    * Returns:
750    *   The allocated data for the given work item.
751    *
752    *****************************************************************/

754   RestoreInfoPtr REST_CreateWorkItemInfo (GREST_Object   object,
755                                           RestoreInfoPtr parent)
756 1 {
757 1   RestoreInfoPtr     newInfo;
                           /* New info created for the WI Object */
758 1   char               wiType;    /* Type of work item */
759 1   eerrno_ty          eerrno;    /* Error code for failed workitem */

761 1   /* Create the new object */
762 1   newInfo = (RestoreInfoPtr) GUTIL_Malloc (sizeof(RestoreInfoRec));
763 1   newInfo->parent = parent;
764 1   newInfo->children = NULL;
765 1   newInfo->next = NULL;

767 1   /* Fill in the data fields */
768 1   newInfo->name = esl_strdup (EDMRST_GetObjectFullName (
                                                GREST_Handle, object));
769 1   newInfo->type = REST_WorkItem;
770 1   newInfo->opened = BOOL_FALSE;
771 1   newInfo->marked = BOOL_FALSE;
772 1   newInfo->restoreObject = object;
773 1   newInfo->backupTime = 0;
774 1   newInfo->errorString = NULL;

776 1   /* Determine which icon to use */
777 1   wiType = EDMRST_GetWorkItemType (GREST_Handle, object);
778 1   if (wiType == WI_TYPE_OFFLINEDB)
779 2   {
780 2     newInfo->icon = REST_DBWorkItemIcon;
781 1   }
782 1   else
783 2   {
784 2     newInfo->icon = REST_FSWorkItemIcon;
785 1   }

787 1   if (!REST_ValidateWorkItem (object, &eerrno))
788 2   {
789 2     newInfo->type = REST_FailedWorkItem;
790 2     newInfo->icon = REST_FailedWorkItemIcon;
791 2     newInfo->errorString = esl_strdup (e_get_error_text(eerrno));
792 2     newInfo->children = REST_CreateErrorInfo (
                                        newInfo->errorString, newInfo);
793 1   }

795 1   newInfo->status = Backup_Good;

797 1   /* Return the new object */
798 1   return (newInfo);
799   }
```

```
801  /*****************************************************************
802   * REST_CreateDirectoryInfo
803   *
804   * Description:
805   *   This routine will create the data for the given directory object.
806   *
807   * Parameters:
808   *   object (I) - The directory object.
809   *   parent (I) - The parent of the new object
810   *
811   * Returns:
812   *   The allocated data for the given directory.
813   *
814   *****************************************************************/

816  RestoreInfoPtr REST_CreateDirectoryInfo (GREST_Object   object,
817                                           RestoreInfoPtr parent)
818 1 {
819 1   RestoreInfoPtr   newInfo;   /* New info created for the Object */

821 1   /* Create the new object */
822 1   newInfo = (RestoreInfoPtr) GUTIL_Malloc (sizeof(RestoreInfoRec));
823 1   newInfo->parent = parent;
824 1   newInfo->children = NULL;
825 1   newInfo->next = NULL;

827 1   /* Fill in the data fields */
828 1   newInfo->name = esl_strdup (EDMRST_GetObjectBaseName (
                                                  GREST_Handle, object));
829 1   newInfo->type = REST_Directory;
830 1   newInfo->icon = REST_DirClosedIcon;
831 1   newInfo->opened = BOOL_FALSE;
832 1   newInfo->marked = GREST_IsObjectMarked (GREST_Handle, object);
833 1   newInfo->restoreObject = object;
834 1   newInfo->status = EDMRST_GetObjectStatus (GREST_Handle, object);
835 1   newInfo->backupTime = 0;
836 1   newInfo->errorString = NULL;

838 1   /* Return the new object */
839 1   return (newInfo);
840  }
```



```
842  /*****************************************************************
843   * REST_CreateFileInfo
844   *
845   * Description:
846   *   This routine will create the data for the given File object.
847   *
848   * Parameters:
849   *   object (I) - The File object.
850   *   parent (I) - The parent of the new object
851   *
852   * Returns:
853   *   The allocated data for the given File.
854   *
855   *****************************************************************/

857  RestoreInfoPtr  REST_CreateFileInfo (GREST_Object   object,
858                                       RestoreInfoPtr parent)
859 1 {
860 1   RestoreInfoPtr   newInfo;   /* New info created for the Object */

862 1   /* Create the new object */
863 1   newInfo = (RestoreInfoPtr) GUTIL_Malloc (sizeof(RestoreInfoRec));
864 1   newInfo->parent = parent;
865 1   newInfo->children = NULL;
866 1   newInfo->next = NULL;

868 1   /* Fill in the data fields */
869 1   newInfo->name = esl_strdup (EDMRST_GetObjectBaseName (
                                                  GREST_Handle, object));
870 1   newInfo->type = REST_File;
871 1   newInfo->icon = REST_FileIcon;
872 1   newInfo->opened = BOOL_FALSE;
873 1   newInfo->restoreObject = object;
874 1   newInfo->status = EDMRST_GetObjectStatus (GREST_Handle, object);
875 1   newInfo->marked = GREST_IsObjectMarked (GREST_Handle, object);
876 1   newInfo->backupTime = 0;
877 1   newInfo->errorString = NULL;

879 1   /* Return the new object */
880 1   return (newInfo);
881  }
```

```
883  /*****************************************************************
884   * REST_AddChild
885   *
886   * Description:
887   *   This routine will add a child to a parent in a non-sorted order.
888   *
889   * Parameters:
890   *   parent (I) - The parent object
891   *   child  (I) - The new child object
892   *
893   * Returns:
894   *   None.
895   *
896   *****************************************************************/

898  void REST_AddChild (RestoreInfoPtr parent,
899                      RestoreInfoPtr child)
900 1 {
901 1   RestoreInfoPtr tmpInfo;
                                  /* Pointer used to walk the children */

903 1   if ((parent != NULL) && (child != NULL))
904 2   {
905 2     /* Add the new child to the list */
906 2     child->next = parent->children;
907 2     parent->children = child;
908 1   }
909   }
```



```
911  /*****************************************************************
912   * REST_CopyInfo
913   *
914   * Description:
915   *   This routine copy the info from the source object to the
                                                          destination.
916   *   If the source info is the current work-item, update the current
917   *   work-item to the destination.
918   *
919   * Parameters:
920   *   sourceInfo (I)      - The object to copy from
921   *   destinationInfo (I) - The object to copy to
922   *
923   * Returns:
924   *   None.
925   *
926   *****************************************************************/

928  void REST_CopyInfo (RestoreInfoPtr sourceInfo,
929                      RestoreInfoPtr destinationInfo)
930 1 {

932 1   if ((sourceInfo != NULL) && (destinationInfo != NULL))
933 2   {

935 2     /* NULL out the links */
936 2     destinationInfo->parent = NULL;
937 2     destinationInfo->children = NULL;
938 2     destinationInfo->next = NULL;

940 2     /* Copy each data field */
941 2     destinationInfo->name          = esl_strdup (sourceInfo->name);
942 2     destinationInfo->type          = sourceInfo->type;
943 2     destinationInfo->icon          = sourceInfo->icon;
944 2     destinationInfo->opened        = sourceInfo->opened;
945 2     destinationInfo->restoreObject = sourceInfo->restoreObject;
946 2     destinationInfo->status        = sourceInfo->status;
947 2     destinationInfo->backupTime    = sourceInfo->backupTime;

949 2     /* If this was the current WI, set it to the new version */
950 2     if (currentWorkItemInfo == sourceInfo)
951 2       currentWorkItemInfo = destinationInfo;
952 1   }
953   }
```

```
955  /*****************************************************************
956   * REST_InitWorkItem
957   *
958   * Description:
959   *   This routine will initialize a work-item.  This can be used so that
960   *   the Restore API will consider the work-item the current work-item
961   *   and the work-item routines can be called.
962   *
963   * Parameters:
964   *   workItemObject (I) - The work-item object to initialize.
965   *   errorCode      (O) - The error code received if we can't init.
966   *
967   * Returns:
968   *   BOOL_TRUE  - If we successfully inited the work-item.
969   *   BOOL_FALSE - If we unsuccessfully inited the work-item.
970   *
971   *****************************************************************/

973  BoolEnum REST_InitWorkItem (GREST_Object  workItemObject,
974                              eerrno_ty    *errorCode)
975 1{
976 1  GREST_Object objects[1];           /* Bogus buffer to get objects */
977 1  long         cookie = INIT_COOKIE; /* Ah, the magic cookie */
978 1  long         numEntries = 0;
                                         /* Bogus value for object count */
979 1  BoolEnum     init = BOOL_FALSE;
                                       /* Flag if the init was sucessful */

981 1  if (workItemObject != NULL)
982 2  {

984 2    /* Create one object */
985 2    if (EDMRST_AllocRestorableObjects (
                                  GREST_Handle, objects, 1) == E_SUCCESS)

986 3    {
987 3      /* Attempt to get the object */
988 3      if ((*errorCode = EDMRST_GetRestorableObjects (GREST_Handle,
989 3                                                     workItemObject,
990 3                                                     BOOL_TRUE,
991 3                                                     1,
992 3                                                     objects,
993 3                                                     &numEntries,
994 3                                                     &cookie)) ==
                                                            E_SUCCESS)

995 4      {
996 4        /* Successfully inited the work-item */
997 4        init = BOOL_TRUE;
998 3      }

1000 3     /* Free up the object */
1001 3     EDMRST_FreeRestorableObjects (GREST_Handle, objects, 1);
1002 2   }
1003 1 }

1005 1 /* Return whether or not we successfully inited the work-item */
1006 1 return (init);
1007  }
```



```
1009 /*****************************************************************
1010  * REST_GetMostRecentWITime
1011  *
1012  * Description:
1013  *   This routine will get the most recent backup time for the given
1014  *   work-item in the given template with the given trail.
1015  *
1016  * Parameters:
1017  *   workItemObject (I) - The work-item object to get the time for.
1018  *   template       (I) - The template to use.
1019  *   isAlternate    (I) - Flag whether or not to use the alternate trail
1020  *
1021  * Returns:
1022  *   The time of the most recent backup, 0 if no backups exist.
1023  *
1024  *****************************************************************/

1026 static time_t REST_GetMostRecentWITime (
                                      GREST_Object        workItemObject,
1027                                   template_name_ty    template,
1028                                   BoolEnum
                                                          isAlternate)
1029 1{
1030 1  BoolEnum     exists;       /* Flag if this template/trail exists */
1031 1  eerrno_ty    eerrno;       /* Error Status */
1032 1  time_t       thisTime = 0; /* Most recent time for work-item */
1033 1  u_long       flags;

1035 1  if (TBUT_GetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton))
1036 1    flags = BACKUP_SELECTION_FLAG_PARTIAL_OK;
1037 1  else
1038 1    flags = BACKUP_SELECTION_FLAG_COMPLETE_ONLY;

1040 1  /* If this is the alternate trail */
1041 1  if (isAlternate)
1042 2  {
1043 2    /* Check if there is an alternate trail for this template */
1044 2    EDMRST_DoesAlternateExist (
                        GREST_Handle, workItemObject, template, &exists);
1045 1  }
1046 1  else
1047 2  {
1048 2    /* Primary trails always exist */
1049 2    exists = BOOL_TRUE;
1050 1  }

1052 1  /* Set to this template using the given trailset */
1053 1  if ((exists) && (EDMRST_SetTopLevelTemplate (GREST_Handle,
1054 1                                               workItemObject,
1055 1                                               template,
1056 1                                               isAlternate) ==
                                                          E_SUCCESS))
1057 2  {

1059 2    /* Initialize the work-item */
1060 2    if (REST_InitWorkItem (workItemObject, &eerrno))
1061 3    {

1063 3      /* Get the most recent backup */
1064 3      if (EDMRST_SetMostRecentBackup (
                                    GREST_Handle, flags) == E_SUCCESS)
1065 4      {

1067 4        /* Initialize the work-item for this time */
```

```
1068  4          if (REST_InitWorkItem (workItemObject, &eerrno))
1069  5          {

1071  5            /* Get the time for this backup */
1072  5            if (EDMRST_GetCurrentBackupTime(
                                    GREST_Handle, &thisTime) != E_SUCCESS)
1073  6            {
1074  6              thisTime = 0;
1075  5            }
1076  4          }
1077  3        }
1078  2      }
1079  1    }

1081  1    return (thisTime);
1082     }
```



```
1084     /*****************************************************************
1085      * REST_GetMostRecentWI
1086      *
1087      * Description:
1088      *   This routine will set the template and trail for the given
                                                                work-item
1089      *   to the most recent.
1090      *
1091      * Parameters:
1092      *   workItemObject (I) - The work-item object to initialize.
1093      *
1094      * Returns:
1095      *   None.
1096      *
1097      *****************************************************************/

1099     static void REST_GetMostRecentWI (GREST_Object workItemObject)
1100  1  {
1101  1    long               cookie = INIT_COOKIE;/* Ah, the magic cookie */
1102  1    short              numTemplates;
                                        /* Number of templates returned */
1103  1    template_name_ty   origTemplate;
                                       /* Original template to use */
1104  1    BoolEnum           origIsAlternate;
                                     /* Original alternate flag to use */
1105  1    BoolEnum           origValid = BOOL_FALSE;
                                      /* Flag if original is valid */
1106  1    template_name_ty   currentTemplate;
                                       /* Current template to use */
1107  1    BoolEnum           currentIsAlternate;
                                     /* Current alternate flag to use */
1108  1    time_t             thisTime;
                                     /* Current time for work-item */
1109  1    time_t             mostRecentTime = 0;
                                     /* Most recent time found so far */
1110  1    template_name_ty   templates[TEMPLATE_BUFFER_LENGTH];
                                          /* Templates returned */
1111  1    Int                i;                 /* Loop Counter */
1112  1    eerrno_ty          eerrno;            /* Error code */
1113  1    u_long             flags;

1115  1    if (TBUT_GetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton))
1116  1      flags = BACKUP_SELECTION_FLAG_PARTIAL_OK;
1117  1    else
1118  1      flags = BACKUP_SELECTION_FLAG_COMPLETE_ONLY;

1120  1    if (REST_InitWorkItem (workItemObject, &eerrno))
1121  2    {
1122  2      if (EDMRST_GetCurrentTemplate (GREST_Handle,
1123  2                                     origTemplate,
1124  2                                     &origIsAlternate) == E_SUCCESS)
1125  2        origValid = BOOL_TRUE;
1126  1    }

1128  1    /* Keep looping while more templates exist */
1129  1    while (cookie != DONE_COOKIE)
1130  2    {
1131  2      /* Get the next group of templates */
1132  2      numTemplates = 0;
1133  2      if (EDMRST_GetTopLevelTemplates (GREST_Handle,
1134  2                                       workItemObject,
1135  2                                       TEMPLATE_BUFFER_LENGTH,
1136  2                                       templates,
1137  2                                       &numTemplates,
1138  2                                       &cookie) == E_SUCCESS)
```

```
1139 3      {
1141 3        /* Loop through each template */
1142 3        for (i=0; i<numTemplates; i++)
1143 4        {
1145 4          /* Get the most recent time for the primary trail */
1146 4          thisTime = REST_GetMostRecentWITime (workItemObject,
1147 4                                               templates[i],
1148 4                                               BOOL_FALSE);
1150 4          /* If this time is more recent than what we have so far */
1151 4          if (thisTime > mostRecentTime)
1152 5          {
1154 5            /* Mark this as the most recent backup */
1155 5            mostRecentTime = thisTime;
1156 5            STR_Cpy (currentTemplate, templates[i]);
1157 5            currentIsAlternate = BOOL_FALSE;
1158 4          }
1160 4          /* Get the most recent time for the alternate trail */
1161 4          thisTime = REST_GetMostRecentWITime (workItemObject,
1162 4                                               templates[i],
1163 4                                               BOOL_TRUE);
1165 4          /* If this time is more recent than what we have so far */
1166 4          if (thisTime > mostRecentTime)
1167 5          {
1169 5            /* Mark this as the most recent backup */
1170 5            mostRecentTime = thisTime;
1171 5            STR_Cpy (currentTemplate, templates[i]);
1172 5            currentIsAlternate = BOOL_TRUE;
1173 4          }
1174 3        }
1175 2      }
1176 2      else
1177 3      {
1178 3        /* Just get out */
1179 3        cookie = DONE_COOKIE;
1180 2      }
1181 1    }
1183 1    /* If we don't have a most recent time and the original was valid */
1184 1    if ((mostRecentTime == 0) && origValid)
1185 2    {
1186 2      /* Get the most recent time for the original template/trail */
1187 2      mostRecentTime = REST_GetMostRecentWITime (workItemObject,
1188 2                                                 origTemplate,
1189 2                                                 origIsAlternate);
1191 2      /* Set the current to the original */
1192 2      STR_Cpy (currentTemplate, origTemplate);
1193 2      currentIsAlternate = origIsAlternate;
1194 1    }
1197 1    /* If we found any templates */
1198 1    if (mostRecentTime != 0)
1199 2    {
1200 2      /* Set to the most recent template and trail */
1201 2      if (EDMRST_SetTopLevelTemplate (GREST_Handle,
1202 2                                      workItemObject,
1203 2                                      currentTemplate,
1204 2                                      currentIsAlternate) == E_SUCCESS)
```



```
1205 3      {
1206 3        /* Initialize this work-item */
1207 3        if (REST_InitWorkItem (workItemObject, &eerrno))
1208 4        {
1209 4          /* Set to the most recent time */
1210 4          EDMRST_SetBackupForTime (GREST_Handle, mostRecentTime, flags);
1211 3        }
1212 2      }
1213 1    }
1215     }
```

```
1217  /*********************************************************************
1218   * REST_ValidateWorkItem
1219   *
1220   * Description:
1221   *   This routine will determine if there are any valid backups for
                                                                        the
1222   *   given workitem.
1223   *
1224   * Parameters:
1225   *   workItemObject (I) - The work-item object to initialize.
1226   *   errorCode      (O) - The error code received if we can't init.
1227   *
1228   * Returns:
1229   *   BOOL_TRUE  - If we found a valid workitem
1230   *   BOOL_FALSE - If we did not find a valid workitem
1231   *
1232   *********************************************************************/

1234  static BoolEnum REST_ValidateWorkItem (GREST_Object  workItemObject,
1235                                         eerrno_ty    *errorCode)
1236  1 {
1237  1   long                cookie = INIT_COOKIE;
                                              /* Ah, the magic cookie */
1238  1   short               numTemplates;
                                          /* Number of templates returned */
1239  1   template_name_ty    templates[TEMPLATE_BUFFER_LENGTH];
                                           /* Templates returned */
1240  1   Int                 i;                        /* Loop Counter */
1241  1   eerrno_ty           eerrno;                   /* Error code */
1242  1   BoolEnum            exists;
                                        /* Flag if alternate exists */
1243  1   BoolEnum            returnVal = BOOL_FALSE;
                                       /* Flag if we found a valid one */

1245  1   /* Try a simple init */
1246  1   if (REST_InitWorkItem (workItemObject, errorCode))
1247  2   {
1248  2     /* got one, wer'e done */
1249  2     returnVal = BOOL_TRUE;
1250  1   }

1252  1   /* Until we find a valid template,
                                 Keep looping while more templates exist */
1253  1   while ((returnVal == BOOL_FALSE) && (cookie != DONE_COOKIE))
1254  2   {
1255  2     /* Get the next group of templates */
1256  2     numTemplates = 0;
1257  2     if (EDMRST_GetTopLevelTemplates (GREST_Handle,
1258  2                                      workItemObject,
1259  2                                      TEMPLATE_BUFFER_LENGTH,
1260  2                                      templates,
1261  2                                      &numTemplates,
1262  2                                      &cookie) == E_SUCCESS)
1263  3     {

1265  3       /* Loop through each template */
1266  3       for (i=0; (i<numTemplates) && (returnVal == BOOL_FALSE); i++)
1267  4       {

1269  4         /* Set to this template using the given trailset */
1270  4         if (EDMRST_SetTopLevelTemplate (GREST_Handle,
1271  4                                         workItemObject,
1272  4                                         templates[i],
1273  4                                         BOOL_FALSE) == E_SUCCESS)
1274  5         {
```



```
1275  5           returnVal = REST_InitWorkItem (workItemObject, &eerrno);
1276  4         }

1278  4         if (returnVal == BOOL_FALSE)
1279  5         {
1280  5           /* Check if there is an alternate trail for this template */
1281  5           EDMRST_DoesAlternateExist (GREST_Handle,
1282  5                                      workItemObject,
1283  5                                      templates[i],
1284  5                                      &exists);

1286  5           if (exists)
1287  6           {
1288  6             if (EDMRST_SetTopLevelTemplate (GREST_Handle,
1289  6                                             workItemObject,
1290  6                                             templates[i],
1291  6                                             BOOL_TRUE) == E_SUCCESS)
1292  7             {
1293  7               returnVal = REST_InitWorkItem (workItemObject, &eerrno);
1294  6             }
1295  5           }
1296  4         }
1297  3       }
1298  2     }
1299  2     else
1300  3     {
1301  3       /* Just get out */
1302  3       cookie = DONE_COOKIE;
1303  2     }
1304  1   }

1306  1   return (returnVal);
1307    }
```

```
1309  /*****************************************************************
1310   * REST_AddWorkItems
1311   *
1312   * Description:
1313   *   This routine will add the work-items for the given parent.
1314   *
1315   * Parameters:
1316   *   parent (I) - The parent object to add work-items for
1317   *
1318   * Returns:
1319   *   BOOL_TRUE  - If any work-items were added.
1320   *   BOOL_FALSE - If no work-items were added.
1321   *
1322   *****************************************************************/

1324  BoolEnum REST_AddWorkItems (RestoreInfoPtr parent)
1325 1{
1326 1  RestoreInfoPtr      info;                 /* New WI info */
1327 1  BoolEnum            returnValue = BOOL_FALSE;
                                                 /* Flag if WIs exist */
1328 1  GREST_Object        workItems[WORK_ITEM_BUFFER_LENGTH];
                                                   /* Array of WIs */
1329 1  GREST_Object       *unused;               /* Unused WIs in array */
1330 1  long               cookie = INIT_COOKIE; /* Ah, the magic cookie */
1331 1  short              numEntries = 0;       /* Number of WIs found */
1332 1  Int                i;                    /* Loop Counter */
1333 1  char               wiType;
                                                /* Type of the work-item */
1334 1  eerrno_ty          eerrno;               /* Error Status */

1336 1  if (parent != NULL)
1337 2  {

1339 2    /* Get all the work items for the given client */
1340 2    while (cookie != DONE_COOKIE)
1341 3    {
1342 3      /* Allocate space for the next group of work-itmes */
1343 3      if (EDMRST_AllocRestorableObjects (GREST_Handle,
1344 3                                         workItems,
1345 3                                         WORK_ITEM_BUFFER_LENGTH) ==
                                                          E_SUCCESS)
1346 4      {

1348 4        /* Get the work items */
1349 4        numEntries = 0;
1350 4        unused = workItems;
1351 4        if ((eerrno = EDMRST_GetTopLevelObjects (GREST_Handle,
1352 4                                                 parent->name,
1353 4
                                                   WORK_ITEM_BUFFER_LENGTH,
1354 4                                                 workItems,
1355 4                                                 &numEntries,
1356 4                                                 &cookie)) ==
                                                          E_SUCCESS)
1357 5        {
1358 5          /* Loop through the returned work items */
1359 5          for (i=0; i<numEntries; i++)
1360 6          {

1362 6            /* Get the work-item type */
1363 6            wiType = EDMRST_GetWorkItemType (
                                         GREST_Handle, workItems[i]);

1365 6            /* If this is a listener work-item, don't show the user */
1366 6            if ((wiType ==  WI_TYPE_OLDB_KICKER) ||
```



```
1367 6                (wiType == WI_TYPE_OLDB_LISTENER))
1368 7            {
1369 7              /* We don't do any restorals on kicker or listener work
                                                              items */
1370 7              EDMRST_FreeRestorableObjects (GREST_Handle, &(
                                                    workItems[i]), 1);
1371 6            }

1373 6            /* Else this must be your everyday work-item */
1374 6            else
1375 7            {
1376 7              /* Create the work-item object */
1377 7              info = REST_CreateWorkItemInfo (workItems[i], parent);

1379 7              /* Add the work-item object to the children list */
1380 7              REST_AddChild (parent, info);
                                              /* Name of the work-item */
1381 6            }

1383 6            /* Bump the unused pointer past this one */
1384 6            unused++;
1385 5          }
1386 4        }
1387 4        else
1388 5        {
1389 5          /* We got an error, ignore it and get out */
1390 5          cookie = DONE_COOKIE;
1391 4        }

1393 4        /* Free up the left overs */
1394 4        if (numEntries < WORK_ITEM_BUFFER_LENGTH)
1395 5        {
1396 5          EDMRST_FreeRestorableObjects (GREST_Handle,
1397 5                                        unused,
1398 5                                        WORK_ITEM_BUFFER_LENGTH -
                                                          numEntries);
1399 4        }
1400 3      }
1401 3      else
1402 4      {
1403 4        /* We got an error, ignore it and get out */
1404 4        cookie = DONE_COOKIE;
1405 3      }
1406 2    }

1408 2    /* Check if we added any work-items */
1409 2    returnValue = (returnValue);

1411 2    /* Now that we have all the children, sort them */
1412 2    REST_SortChildren (parent);
1413 1  }

1415 1  /* Return if we added any children */
1416 1  return (returnValue);
1417  }
```

```
1419  /*****************************************************************
1420   * REST_ValidateClient
1421   *
1422   * Description:
1423   *   This routine will verify a host has something that can be
                                                                  restored
1424   *
1425   * Parameters:
1426   *   clientName (I) - The name of the client to validate
1427   *
1428   * Returns:
1429   *   BOOL_TRUE  - If any work-items can be restored
1430   *   BOOL_FALSE - If no work-items can be restored
1431   *
1432   *****************************************************************/

1434  BoolEnum REST_ValidateClient (Str clientName)
1435 1{
1436 1  BoolEnum             isValid = BOOL_FALSE; /* Flag if WIs exist */
1437 1  GREST_Object         workItems[WORK_ITEM_BUFFER_LENGTH];
                                                    /* Array of WIs */
1438 1  long                 cookie = INIT_COOKIE; /* Ah, the magic cookie */
1439 1  short                numEntries;           /* Number of WIs found */
1440 1  Int                  i;                    /* Loop Counter */
1441 1  char                 wiType;
                                                  /* Type of the work-item */
1442 1  eerrno_ty            eerrno;               /* Error code */

1444 1  if (clientName != NULL)
1445 2  {

1447 2    /* Keep getting workitems for the client until we find a valid one
                                                                        */
1448 2    while ((cookie != DONE_COOKIE) && (isValid == BOOL_FALSE))
1449 3    {
1450 3      /* Allocate space for the next group of work-itmes */
1451 3      if (EDMRST_AllocRestorableObjects (GREST_Handle,
1452 3                                         workItems,
1453 3                                         WORK_ITEM_BUFFER_LENGTH) ==
                                                          E_SUCCESS)
1454 4      {

1456 4        /* Get the work items */
1457 4        numEntries = 0;
1458 4        if (EDMRST_GetTopLevelObjects (GREST_Handle,
1459 4                                       clientName,
1460 4                                       WORK_ITEM_BUFFER_LENGTH,
1461 4                                       workItems,
1462 4                                       &numEntries,
1463 4                                       &cookie) == E_SUCCESS)
1464 5        {
1465 5          /* Loop through the returned work items */
1466 5          for (i=0; (i<numEntries) && (isValid == BOOL_FALSE); i++)
1467 6          {

1469 6            /* Get the work-item type */
1470 6            wiType = EDMRST_GetWorkItemType (
                                          GREST_Handle, workItems[i]);

1472 6            /* If this is not kicker or listener work-item,
                                                          validate it */
1473 6            if ((wiType != WI_TYPE_OLDB_KICKER) &&
1474 6                (wiType != WI_TYPE_OLDB_LISTENER))
1475 7            {
1476 7              /* Attempt to init the workitem,
```



```
                                                  if successful we found one */
1477 7              if (REST_ValidateWorkItem (workItems[i], &eerrno))
1478 8              {
1479 8                isValid = BOOL_TRUE;
1480 7              }
1481 6            }

1483 5          }
1484 4        }
1485 4        else
1486 5        {
1487 5          /* We got an error, ignore it and get out */
1488 5          cookie = DONE_COOKIE;
1489 4        }

1491 4        /* Free up the objects */
1492 4        EDMRST_FreeRestorableObjects (GREST_Handle,
1493 4                                      workItems,
1494 4                                      WORK_ITEM_BUFFER_LENGTH);
1495 3      }
1496 3      else
1497 4      {
1498 4        /* We got an error, ignore it and get out */
1499 4        cookie = DONE_COOKIE;
1500 3      }
1501 2    }
1502 1  }

1504 1  /* Return whether or not we found something restorable */
1505 1  return (isValid);
1506  }
```

```
1508  /*****************************************************************
1509   * REST_CheckForFillCancel
1510   *
1511   * Description:
1512   *   This routine will determine if the user has cancelled the fill
                                                                    and
1513   *   will update the progress window.
1514   *
1515   * Parameters:
1516   *   None.
1517   *
1518   * Returns:
1519   *   BOOL_TRUE  - If the user has cancelled
1520   *   BOOL_FALSE - otherwise
1521   *
1522   *****************************************************************/

1524  static BoolEnum REST_CheckForFillCancel (Int totalItems)
1525 1 {
1526 1   Char        outputString[MAX_STRING_LENGTH];
                                      /* Updated string to display */
1527 1   BoolEnum    retVal;                       /* Value to return */

1529 1   if ((synchFillHandle == NULL) && (
                                    totalItems >= STATUS_REPORT_COUNT))
1530 2   {
1531 2     /* Build the new string */
1532 2     STR_Sprintf (outputString,
1533 2                  REST_GetErrorString (REST_FILL_STATUS),
1534 2                  totalItems);
1535 2     /* Initialize the window */
1536 2     synchFillHandle = GALERT_DisplaySynchronousWait
1537 2                         ((WinPtr)REST_RestoreWin,
1538 2                          REST_GetErrorString (
                                        REST_FILL_PROGRESS_TITLE),
1539 2                          GICON_GetIcon(I_WAIT),
1540 2                          outputString,
1541 2                          BOOL_TRUE);
1542 2     EVENT_Update ();
1543 1   }

1545 1   /* If the number of entries found has changed, update the string */
1546 1   if ((totalItems > 0) && ((totalItems % STATUS_REPORT_COUNT) == 0))
1547 2   {
1548 2     /* Build the new string */
1549 2     STR_Sprintf (outputString,
1550 2                  REST_GetErrorString (REST_FILL_STATUS),
1551 2                  totalItems);

1553 2     /* Update the progress dialog */
1554 2     GALERT_UpdateMessage (synchFillHandle, outputString);

1556 1   }

1558 1   /* Return whether or not the user cancelled */
1559 1   if (synchFillHandle != NULL)
1560 1     retVal = GALERT_IsCancelled(synchFillHandle);
1561 1   else
1562 1     retVal = BOOL_FALSE;

1564 1   return (retVal);
1565  }
```



```
1567  /*****************************************************************
1568   * REST_AddRestorableObjects
1569   *
1570   * Description:
1571   *   This routine will add restorable objects for the given parent
                                                                  object.
1572   *
1573   * Parameters:
1574   *   parent (I) - The parent object to get the children of.
1575   *
1576   * Returns:
1577   *   BOOL_TRUE  - If any children were added.
1578   *   BOOL_FALSE - If no children were added.
1579   *
1580   *****************************************************************/

1582  BoolEnum REST_AddRestorableObjects (RestoreInfoPtr parent)
1583 1 {
1584 1   RestoreInfoPtr  info;                       /* New child info */
1585 1   BoolEnum        returnValue = BOOL_FALSE; /* Value to return */
1586 1   GREST_Object    objects[OBJECTS_BUFFER_LENGTH];
                                                  /* Objects returned */
1587 1   GREST_Object   *unused;
                                        /* Pointer to unused objects */
1588 1   long            cookie = INIT_COOKIE;     /* Ah, the magic cookie */
1589 1   long            numEntries;
                                      /* Number of objects returned */
1590 1   Int             i;                        /* Loop counter */
1591 1   Int             totalCount = 0;
                                    /* The total number found so far */
1592 1   eerrno_ty       eerrno;                   /* Error status */

1594 1   if (parent != NULL)
1595 2   {

1597 2     /* If we are not updating the date,
                                          get the most rescent work-item */
1598 2     if ((parent->type == REST_WorkItem) && (!updatingDate))
1599 3     {
1600 3       REST_GetMostRecentWI (parent->restoreObject);
1601 2     }

1603 2     /* Initialize the fill handle */
1604 2     synchFillHandle = NULL;

1606 2     /* Get all the work items for the given client */
1607 2     while (!REST_CheckForFillCancel (totalCount) && (
                                                 cookie != DONE_COOKIE))
1608 3     {

1610 3       /* Create the next group of objects */
1611 3       if (EDMRST_AllocRestorableObjects (GREST_Handle,
1612 3                                          objects,
1613 3                                          OBJECTS_BUFFER_LENGTH) ==
                                                            E_SUCCESS)
1614 4       {

1616 4         /* Retrieve the next group of objects */
1617 4         numEntries = 0;
1618 4         unused = objects;
1619 4         if ((eerrno = EDMRST_GetRestorableObjects (GREST_Handle,
1620 4
                                                  parent->restoreObject,
1621 4                                             BOOL_TRUE,
1622 4
```

```
                                                    OBJECTS_BUFFER_LENGTH,
1623  4                                                 objects,
1624  4                                                 &numEntries,
1625  4                                                 &cookie)) ==
                                                              E_SUCCESS)
1626  5         {

1628  5           /* Loop through all objects */
1629  5           for (i=0; i<numEntries; i++)
1630  6           {

1632  6             /* If this is a directory create the directory object */
1633  6             if (EDMRST_IsObjectContainer (GREST_Handle, objects[i]))
1634  7             {
1635  7               info = REST_CreateDirectoryInfo (objects[i], parent);
1636  6             }

1638  6             /* If this is a file create the directory object */
1639  6             else if (EDMRST_IsObjectLeaf (GREST_Handle, objects[i]))
1640  7             {
1641  7               info = REST_CreateFileInfo (objects[i], parent);
1642  6             }

1644  6             /* Else this is an unknown object, treat it like a file */
1645  6             else
1646  7             {
1647  7               info = REST_CreateFileInfo (objects[i], parent);
1648  6             }

1650  6             /* Add this child to the parent */
1651  6             REST_AddChild (parent, info);
1652  6             totalCount++;
1653  6             returnValue = BOOL_TRUE;

1655  6             /* Bump the unused pointer past this one */
1656  6             unused++;
1657  5           }
1658  4         }
1659  4         else
1660  5         {
1661  5           /*
1662  5            * If this is not a re-read, display an error
1663  5            *   (else it would be displayed twice)
1664  5            */

1666  5           if (!REST_IsReReadInProgress())
1667  6           {
1668  6             Char outputString[MAX_STRING_LENGTH];
                                                  /* String to display */

1670  6             /* Get the message to display */
1671  6             STR_Sprintf (outputString,
1672  6                          REST_GetErrorString (REST_OPEN_OBJECT_ERROR),
1673  6                          EDMRST_GetObjectFullName (
                                       GREST_Handle, parent->restoreObject));

1675  6             /* display the error message */
1676  6             REST_DisplayErrnoMessage ((WinPtr)REST_RestoreWin,
1677  6                                       NULL,
1678  6                                       outputString,
1679  6                                       eerrno);
1680  5           }

1682  5           /* Get out of the loop */
1683  5           cookie = DONE_COOKIE;
```



```
1685  5           /*
1686  5            * If this was a workitem, flag it as failed and get out
1687  5            */

1689  5           if (parent->type == REST_WorkItem)
1690  6           {
1691  6             parent->type = REST_FailedWorkItem;
1692  6             parent->errorString = esl_strdup (e_get_error_text(
                                                              eerrno));
1693  6             parent->children = REST_CreateErrorInfo (
                                                    parent->errorString,
1694  6                                                 parent);
1695  5           }
1696  4         }

1698  4         /* Free up the left overs */
1699  4         if (numEntries < OBJECTS_BUFFER_LENGTH)
1700  5         {
1701  5           EDMRST_FreeRestorableObjects (GREST_Handle,
1702  5                                         unused,
1703  5                                         OBJECTS_BUFFER_LENGTH -
                                                        numEntries);
1704  4         }
1705  3       }
1706  3       else
1707  4       {
1708  4         /* Got an error, ignore it and get out */
1709  4         cookie = DONE_COOKIE;
1710  3       }
1711  2     }

1713  2     /* The fill handle is not NULL, remove the progress window. */
1714  2     if (synchFillHandle != NULL)
1715  3     {
1716  3       GALERT_CancelSynchDialog (synchFillHandle);
1717  3       synchFillHandle = NULL;
1718  2     }

1720  2     /* Now that we have all the children, sort them */
1721  2     REST_SortChildren (parent);
1722  1   }

1724  1   /* Return whether or not we found children */
1725  1   return (returnValue);
1726    }
```

```
1728  /*****************************************************************
1729   * REST_CreateInfoChildren
1730   *
1731   * Description:
1732   *   This routine will create the children for the given object.
1733   *
1734   * Parameters:
1735   *   parent (I) - The parent object to get the children of.
1736   *
1737   * Returns:
1738   *   None.
1739   *
1740   *****************************************************************/

1742  void REST_CreateInfoChildren (RestoreInfoPtr parent)
1743 1 {

1745 1   if (parent != NULL)
1746 2   {

1748 2     /* Call the correct add routine based on type */
1749 2     switch (parent->type)
1750 3     {
1751 3       case REST_Client:
1752 3         REST_AddWorkItems (parent);
1753 3         break;
1754 3       case REST_WorkItem:
1755 3         REST_AddRestorableObjects (parent);
1756 3         break;
1757 3       case REST_Directory:
1758 3         REST_AddRestorableObjects (parent);
1759 3         break;
1760 3       case REST_File:
1761 3         break;
1762 3       default:
1763 3         break;
1764 2     }
1765 1   }
1766  }
```



```
1768  /*****************************************************************
1769   * REST_FindInfoInChildren
1770   *
1771   * Description:
1772   *   This routine will find the info object for the given full
1773   *   child name.
1774   *
1775   * Parameters:
1776   *   itemFullName (I) - The full name of the object to be found.
1777   *   parent       (I) - The parent object to start the search from.
1778   *   getChildren  (
                    I) - Flag if new children should be created if necessary
1779   *
1780   * Returns:
1781   *   The found object or NULL if not found
1782   *
1783   *****************************************************************/

1785  RestoreInfoPtr REST_FindInfoInChildren (Str            itemFullName,
1786                                          RestoreInfoPtr parent,
1787                                          BoolEnum       getChildren)
1788 1 {
1789 1   RestoreInfoPtr tmpInfo;
                                  /* Pointer to walk the children with */
1790 1   RestoreInfoPtr foundInfo = NULL; /* Matching info found */

1792 1   REST_StandardizePath(itemFullName);

1794 1   if (parent != NULL)
1795 2   {

1797 2     /* If there are no children yet, add them (but don't display) */
1798 2     if (getChildren && (parent->children == NULL) && (
                                              parent->type != REST_File))

1799 3     {
1800 3       /* Set the flag so that the objects aren't displayed */
1801 3       updatingDate = BOOL_TRUE;

1803 3       /* Add the child objects */
1804 3       REST_CreateInfoChildren (parent);

1806 3       /* Set the flag back */
1807 3       updatingDate = BOOL_FALSE;
1808 2     }

1810 2     /* Loop through the children */
1811 2     tmpInfo = parent->children;
1812 2     while ((foundInfo == NULL) && (tmpInfo != NULL))
1813 3     {

1815 3       /* Check if this is the one */
1816 3       if (STR_Cmp (itemFullName, REST_GetFullName (
                                                  tmpInfo)) == CMP_EQUAL)
1817 4       {
1818 4         foundInfo = tmpInfo;
1819 3       }

1821 3       /* If this could be the parent, the check its children */
1822 3       else if ((tmpInfo->type != REST_File) &&
1823 3                (REST_IsParentString (tmpInfo, itemFullName)))
1824 4       {
1825 4         foundInfo = REST_FindInfoInChildren(
                                    itemFullName, tmpInfo, getChildren);
1826 3       }
```

```
1828  3         /* Go to the next child */
1829  3         tmpInfo = tmpInfo->next;
1830  2       }
1831  1     }

1833  1     /* Return the found object or NULL if not found */
1834  1     return (foundInfo);
1835    }
```



```
1837    /*****************************************************************
1838     * REST_StartClientList
1839     *
1840     * Description:
1841     *   This routine will initialize the global client list.
                                                                  It will free up
1842     *   the old client list if it existed.
1843     *
1844     * Parameters:
1845     *   None.
1846     *
1847     * Returns:
1848     *   None.
1849     *
1850     *****************************************************************/

1852    void REST_StartClientList (void)
1853  1 {
1854  1     RestoreClientPtr currClient;
                                /* Current client pointer to walk the list */
1855  1     RestoreClientPtr nextClient;  /* Next client in the list */

1857  1     /* If there is already a list, free it up */
1858  1     if (currentClientList != NULL)
1859  2     {
1860  2       /* Start at the beginning of the list */
1861  2       currClient = currentClientList;

1863  2       /* Loop until there are no more clients */
1864  2       while (currClient != NULL)
1865  3       {
1866  3         /* Save the next client pointer */
1867  3         nextClient = currClient->next;

1869  3         /* Free up this client */
1870  3         GUTIL_Free ((VoidPtr)currClient);

1872  3         /* Go to the next client */
1873  3         currClient = nextClient;
1874  2       }

1876  2       /* Set the current Client list to NULL */
1877  2       currentClientList = NULL;

1879  1     }

1881    }
```

```
1883  /*********************************************************************
1884   * REST_AddClient
1885   *
1886   * Description:
1887   *   This routine will add a client to the global client list.
1888   *
1889   * Parameters:
1890   *   clientName (I) - name of the client to add
1891   *
1892   * Returns:
1893   *   None.
1894   *
1895   *********************************************************************/

1897  void REST_AddClient (Str clientName)
1898  1 {
1899  1   RestoreClientPtr newClient;  /* The new client info */
1900  1   RestoreClientPtr nextClient;
                            /* Pointer to walk the current list with */


1902  1   /* Create the new client */
1903  1   newClient = (RestoreClientPtr) GUTIL_Malloc (sizeof(
                                                    RestoreClientRec));

1904  1   STR_Cpy (newClient->clientName, clientName);
1905  1   newClient->next = NULL;

1907  1   /* attach the new client to the end of the list */
1908  1   if (currentClientList != NULL)
1909  2   {
1910  2     nextClient = currentClientList;
1911  2     while (nextClient->next != NULL)
1912  3     {
1913  3       nextClient = nextClient->next;
1914  2     }
1915  2     nextClient->next = newClient;
1916  1   }
1917  1   else
1918  2   {
1919  2     currentClientList = newClient;
1920  1   }
1921  }
```



```
1923  /*********************************************************************
1924   * REST_StartWiList
1925   *
1926   * Description:
1927   *   This routine will initialize the global work item list.
                                                        It will free up
1928   *   the old work item list if it existed.
1929   *
1930   * Parameters:
1931   *   None.
1932   *
1933   * Returns:
1934   *   None.
1935   *
1936   *********************************************************************/

1938  void REST_StartWiList (void)
1939  1 {
1940  1   RestoreWorkItemPtr currWI;
                                  /* Current WI pointer to walk the list */
1941  1   RestoreWorkItemPtr nextWI;  /* Next WI in the list */

1943  1   /* If there is already a list, free it up */
1944  1   if (currentWIList != NULL)
1945  2   {
1946  2     /* Start at the beginning of the list */
1947  2     currWI = currentWIList;

1949  2     /* Loop until there are no more WIs */
1950  2     while (currWI != NULL)
1951  3     {
1952  3       /* Save the next WI pointer */
1953  3       nextWI = currWI->next;

1955  3       /* Free up this WI */
1956  3       GUTIL_Free ((VoidPtr)currWI);

1958  3       /* Go to the next WI */
1959  3       currWI = nextWI;
1960  2     }

1962  2     /* Set the current WI list to NULL */
1963  2     currentWIList = NULL;

1965  1   }

1967  }
```

```
1969  /*****************************************************************
1970   * REST_AddWi
1971   *
1972   * Description:
1973   *   This routine will add a work item to the global work item list.
1974   *
1975   * Parameters:
1976   *   wiName (I) - name of the work item to add
1977   *
1978   * Returns:
1979   *   None.
1980   *
1981   *****************************************************************/

1983  void REST_AddWi (Str wiName)
1984  1 {
1985  1   RestoreWorkItemPtr newWI;  /* The new WI info */
1986  1   RestoreWorkItemPtr nextWI;
                              /* Pointer to walk the current list with */

1988  1   /* Create the new work item */
1989  1   newWI = (RestoreWorkItemPtr) GUTIL_Malloc (sizeof(
                                                    RestoreWorkItemRec));
1990  1   STR_Cpy (newWI->wiName, wiName);
1991  1   newWI->next = NULL;

1993  1   /* attach the new work item to the end of the list */
1994  1   if (currentWIList != NULL)
1995  2   {
1996  2     nextWI = currentWIList;
1997  2     while (nextWI->next != NULL)
1998  3     {
1999  3       nextWI = nextWI->next;
2000  2     }
2001  2     nextWI->next = newWI;
2002  1   }
2003  1   else
2004  2   {
2005  2     currentWIList = newWI;
2006  1   }
2007    }
```



```
2009  /*****************************************************************
2010   * REST_UnmarkProgressCB
2011   *
2012   * Description:
2013   *   This routine will report current unmark progress to the user
2014   *
2015   * Parameters:
2016   *   totalMarks (I) - the number of unmarked items
2017   *
2018   * Returns:
2019   *   BOOL_TRUE  - If the unmark operation should continue
2020   *   BOOL_FALSE - If the user cancelled the operation
2021   *
2022   *****************************************************************/

2024  static BoolEnum REST_UnmarkProgressCB (unsigned long totalMarks)
2025  1 {
2026  1   Char     outputString[MAX_STRING_LENGTH];
                                          /* Message string to display */

2028  1   STR_Sprintf (outputString,
2029  1                REST_GetErrorString (REST_UNMARK_PROGRESS_FORMAT),
2030  1                totalMarks);

2032  1   if (synchMarkHandle == NULL)
2033  2   {
2034  2     /* Initialize the window */
2035  2     synchMarkHandle = GALERT_DisplaySynchronousWait
2036  2                          ((WinPtr)REST_RestoreWin,
2037  2                           REST_GetErrorString (
                                              REST_UNMARK_PROGRESS_TITLE),
2038  2                           GICON_GetIcon(I_WAIT),
2039  2                           outputString,
2040  2                           BOOL_TRUE);

2042  1   }
2043  1   else
2044  2   {
2045  2     GALERT_UpdateMessage (synchMarkHandle, outputString);
2046  1   }

2048  1   /* Return whether or not the user cancelled */
2049  1   return (! GALERT_IsCancelled(synchMarkHandle));
2050    }
```

```
2052  /*********************************************************************
2053   * REST_MarkRestorableObject
2054   *
2055   * Description:
2056   *   This routine mark the given restore object.
2057   *
2058   * Parameters:
2059   *   restoreObject (I) - the object to mark
2060   *   backupTime    (I) - the time of the backup for the object
2061   *   numberMarked  (O) - the number of marked items
2062   *   numberBad     (O) - the number of bad items marked
2063   *
2064   * Returns:
2065   *   BOOL_TRUE  - If the object was marked successfully
2066   *   BOOL_FALSE - otherwise
2067   *
2068   *********************************************************************/

2070  BoolEnum REST_MarkRestorableObject (GREST_Object  restoreObject,
2071                                      time_t        backupTime,
2072                                      long         *numberMarked,
2073                                      long         *numberBad)

2074 1 {
2075 1   BoolEnum   marked = BOOL_FALSE; /* Flag if marking was successful */
2076 1   eerrno_ty  eerrno;              /* Error Status */
2077 1   Char       outputString[MAX_STRING_LENGTH];
                                        /* Error string to display */
2078 1   u_long     badFiles = 0;        /* Number of bad files */
2079 1   u_long     permDeniedFiles = 0;
                                       /* Number of permission denied files */
2080 1   u_long     markedFiles = 0;     /* Number of marked files */
2081 1   u_long     markedDirs = 0;      /* Number of marked directories */
2082 1   u_long     markedOthers = 0;    /* Number of marked other types */
2083 1   u_long     totalMarks = 0;      /* Number of total marks */
2084 1   Str        fullName;            /* Full name of the object */
2085 1   boolean_ty interrupt = BOOL_FALSE; /* Flag to interrupt operation */

2087 1   /* Validate the object */
2088 1   if ((restoreObject != NULL) && (numberMarked != NULL) && (
                                                    numberBad != NULL))
2089 2   {

2091 2     /*
2092 2      * Check if an object by this name is alread selected and warn the
2093 2      * user before marking it.
                                  This can happen if the user attempts to
2094 2      * mark the same file backed up at different times.
2095 2      */

2097 2     fullName = esl_strdup (EDMRST_GetObjectFullName (
                                        GREST_Handle, restoreObject));
2098 2     REST_StripDirectoryChars (fullName);

2100 2     if (REST_IsNameSelected (fullName))
2101 3     {
2102 3       /* Tell the user the object name is already marked */
2103 3       STR_Sprintf (outputString,
2104 3                    REST_GetErrorString (REST_NAME_MARKED_FORMAT),
2105 3                    fullName);

2107 3       /* Ask the user if he/she wants to continue anyways */
2108 3       if (GALERT_DisplayQuestion((WinPtr)REST_RestoreWin,
2109 3                                  REST_GetErrorString (
                                              REST_NAME_MARKED_WARNING),
2110 3                                  GICON_GetWarning(),
```



```
2111 3                                  outputString,
2112 3                                  BOOL_FALSE) != GALERT_Affirmative)
2113 4       {
2114 4         GUTIL_Free (fullName);
2115 4         return (BOOL_FALSE);
2116 3       }

2118 2     }

2120 2     /* Initialize the current mark handle to NULL */
2121 2     synchMarkHandle = NULL;

2123 2     /* If time zero was passed, mark at the current backup time */
2124 2     if (backupTime == 0)
2125 3     {

2127 3       /* Make sure the object is markable */
2128 3       if (!GREST_IsObjectMarkable (GREST_Handle, restoreObject))
2129 4       {

2131 4         /* tell the user it can't be marked */
2132 4         STR_Sprintf (outputString,
2133 4                      REST_GetErrorString (REST_MARK_ERROR),
2134 4                      fullName);
2135 4         /* Display the error message */
2136 4         GALERT_DisplayError ((WinPtr)REST_RestoreWin,
2137 4                              REST_GetErrorString (REST_ERROR_INDEX),
2138 4                              GICON_GetError(),
2139 4                              outputString);

2141 4         /* Return that the object wasn't marked */
2142 4         GUTIL_Free (fullName);
2143 4         return (BOOL_FALSE);
2144 3       }
2145 2     }

2147 2     /* Mark the object */
2148 2     eerrno = EDMRST_MarkObject (GREST_Handle,
2149 2                                 restoreObject,
2150 2                                 backupTime,
2151 2                                 REST_MarkBadFiles,
2152 2                                 BOOL_TRUE);

2154 2     if (eerrno == E_SUCCESS)
2155 3     {
2156 3       while ((eerrno = EDMRST_GetMarkResults (GREST_Handle,
2157 3                                               interrupt,
2158 3                                               &badFiles,
2159 3                                               &permDeniedFiles,
2160 3                                               &markedFiles,
2161 3                                               &markedDirs,
2162 3                                               &markedOthers)) ==
                                            EP_RB_RECOVER_RPC_INCOMPLETE)
2163 4         {
2164 4           totalMarks = markedFiles + markedDirs + markedOthers;
2165 4           if (totalMarks > REST_MARK_THRESHOLD)
2166 5           {
2167 5             STR_Sprintf (outputString,
2168 5                          REST_GetErrorString (
                                          REST_MARK_PROGRESS_FORMAT),
2169 5                          totalMarks);

2171 5             if (synchMarkHandle == NULL)
2172 6             {
2173 6               /* Initialize the window */
2174 6               synchMarkHandle = GALERT_DisplaySynchronousWait
```

```
                                    ((WinPtr)REST_RestoreWin,
                                     REST_GetErrorString (
                                                REST_MARK_PROGRESS_TITLE),
                                     GICON_GetIcon(I_WAIT),
                                     outputString,
                                     BOOL_TRUE);

            }
            else
            {
              GALERT_UpdateMessage (synchMarkHandle, outputString);
            }

            /* Determine if the user cancelled yet */
            interrupt = GALERT_IsCancelled(synchMarkHandle);
          }
        }

        if (synchMarkHandle != NULL)
        {
          if (GALERT_IsCancelled (synchMarkHandle))
          {
            /* Display the warning message */
            GALERT_DisplayError ((WinPtr)REST_RestoreWin,
                                 REST_GetErrorString (
                                            REST_MARK_CANCELLED_TITLE),
                                 GICON_GetWarning(),
                                 REST_GetErrorString (
                                         REST_MARK_CANCELLED_MESSAGE));

          }

          GALERT_CancelSynchDialog (synchMarkHandle);
          synchMarkHandle = NULL;
        }
      }

      /* If there is no error, update the marked data */
      if (eerrno == E_SUCCESS)
      {
        /* Set the marked and bad count */
        *numberMarked = markedFiles + markedDirs + markedOthers;
        if (REST_MarkBadFiles)
        {
          *numberBad = badFiles;
        }
        else
        {
          *numberBad = 0;
        }

        /* Successfully marked,  add the object to the selection list */
        REST_SelectRestorableItem (restoreObject, backupTime);
        marked = BOOL_TRUE;
      }

      /* Else display an error message to the user */
      else
      {
        STR_Sprintf (outputString,
                     REST_GetErrorString (REST_MARK_ERROR),
                     fullName);
        REST_DisplayErrnoMessage ((WinPtr)REST_RestoreWin,
                                  NULL,
                                  outputString,
                                  eerrno);
      }
```



```
    GUTIL_Free (fullName);
  }

  /* Return whether or not the object was marked */
  return (marked);
}
```

```
2246  /*********************************************************************
2247   * REST_UnmarkRestorableObject
2248   *
2249   * Description:
2250   *   This routine unmark the given restore object.
2251   *
2252   * Parameters:
2253   *   restoreObject (I) - the object to mark
2254   *   backupTime    (I) - the time of the backup for the object
2255   *   numberMarked  (O) - the number of marked items
2256   *   numberBad     (O) - the number of bad items marked
2257   *
2258   * Returns:
2259   *   BOOL_TRUE  - If the object was unmarked successfully
2260   *   BOOL_FALSE - otherwise
2261   *
2262   *********************************************************************/

2264  BoolEnum REST_UnmarkRestorableObject (GREST_Object  restoreObject,
2265                                        time_t        backupTime,
2266                                        long         *numberMarked,
2267                                        long         *numberBad)
2268 1{
2269 1  BoolEnum   unmarked = BOOL_FALSE;
                                  /* Flag if marking was successful */
2270 1  Char       outputString[MAX_STRING_LENGTH];
                                     /* Error string to display */
2271 1  eerrno_ty  eerrno;                 /* Error Status */
2272 1  u_long     badFiles = 0;           /* Number of bad files */
2273 1  u_long     markedFiles = 0;        /* Number of marked files */
2274 1  u_long     markedDirs = 0;         /* Number of marked directories */
2275 1  u_long     markedOthers = 0;       /* Number of marked other types */
2276 1  u_long     totalMarks = 0;         /* Number of total marks */
2277 1  boolean_ty interrupt = BOOL_FALSE; /* Flag to interrupt operation */

2279 1  /* Validate the object */
2280 1  if ((restoreObject != NULL) && (numberMarked != NULL) && (
                                                    numberBad != NULL))
2281 2  {
2282 2    /* Initialize the current mark handle to NULL */
2283 2    synchMarkHandle = NULL;

2285 2    eerrno = EDMRST_UnmarkObject (GREST_Handle,
2286 2                                  restoreObject,
2287 2                                  backupTime,
2288 2                                  BOOL_FALSE,
2289 2                                  BOOL_TRUE);
2290 2    if (eerrno == E_SUCCESS)
2291 3    {
2292 3      while ((eerrno = EDMRST_GetUnmarkResults (GREST_Handle,
2293 3                                                interrupt,
2294 3                                                &badFiles,
2295 3                                                &markedFiles,
2296 3                                                &markedDirs,
2297 3                                                &markedOthers)) ==
                                             EP_RB_RECOVER_RPC_INCOMPLETE)
2298 4      {
2299 4        totalMarks = markedFiles + markedDirs + markedOthers;
2300 4        if (totalMarks > REST_MARK_THRESHOLD)

2302 5        {
2303 5          STR_Sprintf (outputString,
2304 5                       REST_GetErrorString (
                                             REST_UNMARK_PROGRESS_FORMAT),
2305 5                       totalMarks);
```



```
2307 5          if (synchMarkHandle == NULL)
2308 6          {
2309 6            /* Initialize the window */
2310 6            synchMarkHandle = GALERT_DisplaySynchronousWait
2311 6                                  ((WinPtr)REST_RestoreWin,
2312 6                                   REST_GetErrorString (
                                               REST_UNMARK_PROGRESS_TITLE),
2313 6                                   GICON_GetIcon(I_WAIT),
2314 6                                   outputString,
2315 6                                   BOOL_TRUE);

2317 5          }
2318 5          else
2319 6          {
2320 6            GALERT_UpdateMessage (synchMarkHandle, outputString);
2321 5          }

2323 5          /* Determine if the user cancelled yet */
2324 5          interrupt = GALERT_IsCancelled(synchMarkHandle);
2325 4        }
2326 3      }

2328 3      if (synchMarkHandle != NULL)
2329 4      {
2330 4        if (GALERT_IsCancelled (synchMarkHandle))
2331 5        {
2332 5          /* Display the warning message */
2333 5          GALERT_DisplayError ((WinPtr)REST_RestoreWin,
2334 5                               REST_GetErrorString (
                                           REST_UNMARK_CANCELLED_TITLE),
2335 5                               GICON_GetWarning(),
2336 5                               REST_GetErrorString (
                                         REST_UNMARK_CANCELLED_MESSAGE));
2337 4        }

2339 4        GALERT_CancelSynchDialog (synchMarkHandle);
2340 4        synchMarkHandle = NULL;
2341 3      }
2342 2    }

2344 2    /* If there is no error, update the marked data */
2345 2    if (eerrno == E_SUCCESS)
2346 3    {
2347 3      unmarked = BOOL_TRUE;
2348 3      *numberMarked = markedFiles + markedDirs + markedOthers;
2349 3      if (REST_MarkBadFiles)
2350 4      {
2351 4        *numberBad = -badFiles;
2352 3      }
2353 3      else
2354 4      {
2355 4        *numberBad = 0;
2356 3      }

2358 3      /* Deselect the object */
2359 3      REST_DeselectInfo (restoreObject, backupTime);
2360 2    }

2362 2    /* Else display an error message to the user */
2363 2    else
2364 3    {
2365 3      Char       outputString[MAX_STRING_LENGTH];
                                          /* Error output string */

2367 3      /* Create the error string */
```

```
2368  3          STR_Sprintf (outputString,
2369  3                       REST_GetErrorString (REST_UNABLE_TO_UNMARK),
2370  3                       EDMRST_GetObjectFullName (
                                          GREST_Handle, restoreObject));

2372  3          /* Display the error message */
2373  3          REST_DisplayErrnoMessage ((WinPtr)REST_RestoreWin,
2374  3                                    NULL,
2375  3                                    outputString,
2376  3                                    eerrno);
2377  2        }
2378  1      }

2380  1      /* Return whether or not the object was unmarked */
2381  1      return (unmarked);
2382    }
```



```
2384    /*****************************************************************
2385     * REST_MarkInfo
2386     *
2387     * Description:
2388     *   This routine will unmark the given info object
2389     *
2390     * Parameters:
2391     *   info          - The info object to unmark
2392     *   numberMarked  - The number of marked objects
2393     *   numberBad     - The number of bad files unmarked
2394     *
2395     * Returns:
2396     *   None.
2397     *
2398     *****************************************************************/

2400    BoolEnum REST_MarkInfo (RestoreInfoPtr  info,
2401                            long          *numberMarked,
2402                            long          *numberBad)
2403  1 {
2404  1   BoolEnum          thisMark = BOOL_FALSE;
                                        /* Flag if this mark was successful */
2405  1   long              thisBad = 0;
                                       /* Number of bad files in a mark */
2406  1   RestoreInfoPtr    nextChild;
                                         /* Loop pointer to next object */

2408  1   if (info != NULL)
2409  2   {
2410  2     /* If this is a file or a directory mark the restorable object */
2411  2     if ((!info->marked) &&
2412  2         ((info->type == REST_File) || (info->type == REST_Directory)))
2413  3     {
2414  3       thisMark = REST_MarkRestorableObject (info->restoreObject,
2415  3                                             0,
2416  3                                             numberMarked,
2417  3                                             numberBad);
2418  2     }
2419  2     else if ((!info->marked) && (info->type == REST_WorkItem))
2420  3     {
2421  3       if (info->children == NULL)
2422  4       {
2423  4         /* Set the flag so that the objects aren't displayed */
2424  4         updatingDate = BOOL_TRUE;

2426  4         /* Add the child objects */
2427  4         REST_CreateInfoChildren (info);

2429  4         /* Set the flag back */
2430  4         updatingDate = BOOL_FALSE;
2431  3       }

2433  3       /* Loop through and mark all the children */
2434  3       nextChild = info->children;
2435  3       while (nextChild != NULL)
2436  4       {

2438  4         /* Mark this child */
2439  4         if (!nextChild->marked)
2440  5         {
2441  5           thisMark |= REST_MarkRestorableObject (
                                                nextChild->restoreObject,
2442  5                                                 0,
2443  5                                                 numberMarked,
2444  5                                                 &thisBad);
```

```
2445  5            /* Add in the number of bad files for this mark */
2446  5            *numberBad = *numberBad + thisBad;
2447  4          }

2449  4          /* Move on to the next child */
2450  4          nextChild = nextChild->next;
2451  3        }
2452  2      }

2454  2      /* Update the mark flags for all objects */
2455  2      REST_UpdateObjectMarks (currentWorkItemInfo);
2456  1    }

2458  1    return (thisMark);
2459      }
```



```
2461  /*********************************************************************
2462   * REST_UnmarkInfo
2463   *
2464   * Description:
2465   *   This routine will unmark the given info object
2466   *
2467   * Parameters:
2468   *   info       - The info object to unmark
2469   *   numberMarked - The number of marked objects
2470   *   numberBad - The number of bad files unmarked
2471   *
2472   * Returns:
2473   *   None.
2474   *
2475   *********************************************************************/

2477  BoolEnum REST_UnmarkInfo (RestoreInfoPtr  info,
2478                            long           *numberMarked,
2479                            long           *numberBad)
2480  1 {
2481  1   BoolEnum       thisMark = BOOL_FALSE;
                                  /* Flag if this mark was successful */
2482  1   eerrno_ty      eerrno;                 /* Error status */
2483  1   time_t         currentBackupTime;
                                    /* Time of the current backup */
2484  1   long           thisBad = 0;
                                  /* Number of bad files in a mark */
2485  1   RestoreInfoPtr nextChild;
                                    /* Loop pointer to next object */

2487  1   if (info != NULL)
2488  2   {
2489  2     /* remove the item from the list */
2490  2     if ((info->marked) &&
2491  2         ((info->type == REST_File) || (info->type == REST_Directory)))
2492  3     {
2493  3       /* Set the backup time to the current backup time */
2494  3       if ((eerrno = EDMRST_GetCurrentBackupTime(
                                  GREST_Handle, &currentBackupTime)) != 0)
2495  4       {
2496  4         /* Hmm... I guess we just use time zero */
2497  4         currentBackupTime = 0;
2498  3       }

2500  3       thisMark = REST_UnmarkRestorableObject (info->restoreObject,
2501  3                                               currentBackupTime,
2502  3                                               numberMarked,
2503  3                                               numberBad);
2504  2     }
2505  2     else if ((info->marked) && (info->type == REST_WorkItem))
2506  3     {
2507  3       /* Set the backup time to the current backup time */
2508  3       if ((eerrno = EDMRST_GetCurrentBackupTime(
                                  GREST_Handle, &currentBackupTime)) != 0)
2509  4       {
2510  4         /* Hmm... I guess we just use time zero */
2511  4         currentBackupTime = 0;
2512  3       }

2514  3       if (info->children == NULL)
2515  4       {
2516  4         /* Set the flag so that the objects aren't displayed */
2517  4         updatingDate = BOOL_TRUE;

2519  4         /* Add the child objects */
```

```
2520  4          REST_CreateInfoChildren (info);

2522  4          /* Set the flag back */
2523  4          updatingDate = BOOL_FALSE;
2524  3        }

2526  3        /* Loop through and mark all the children */
2527  3        nextChild = info->children;
2528  3        while (nextChild != NULL)
2529  4        {
2530  4          /* Mark this child */
2531  4          thisMark = REST_UnmarkRestorableObject (
                                              nextChild->restoreObject,
2532  4                                           currentBackupTime,
2533  4                                           numberMarked,
2534  4                                           &thisBad);
2535  4          /* Add in the number of bad files for this mark */
2536  4          *numberBad = *numberBad + thisBad;

2538  4          /* Move on to the next child */
2539  4          nextChild = nextChild->next;
2540  3        }
2541  2      }

2543  2      /* Update the mark flags for all objects */
2544  2      REST_UpdateObjectMarks (currentWorkItemInfo);
2545  1    }

2547  1    return (thisMark);
2548    }
```



```
2550  /*****************************************************************
2551   * REST_Initialize
2552   *
2553   * Description:
2554   *   This routine will initialize all componets of restore. It will
2555   *   load the window resources and set up all default values.
2556   *
2557   * Parameters:
2558   *   None.
2559   *
2560   * Returns:
2561   *   None.
2562   *
2563   *****************************************************************/

2565  void REST_Initialize (void)
2566  1 {
2567  1    REST_SortType  sortType;
                                                    /* Initial sort type */
2568  1    Str            hostName;
                                                    /* Client string */
2569  1    Char           windowLabel[2 * GMAX_HOSTNAME_LENGTH];
                                                    /* New window label */


2571  1    /* Start off clean */
2572  1    currentWorkItemInfo = NULL;

2574  1    /* Initialize the option flags */
2575  1    REST_ShowHiddenFiles = BOOL_TRUE;
2576  1    REST_ShowBadFiles = BOOL_FALSE;
2577  1    REST_MarkBadFiles = BOOL_FALSE;

2579  1    /* Initialize the File Manager */
2580  1    GFMGR_Initialize ();

2582  1    /* Load the restore window data */
2583  1    REST_RestoreWinLoadInit ();

2585  1    /* Get the string list of trail names */
2586  1    REST_TrailNameList  = (StrLPtr)RES_LoadInit (
                                                  "restore", "TrailNames");


2588  1    /* Set up the tabs */
2589  1    REST_TabObject = GTAB_objInit (REST_RestoreWin->TabsPanel);
2590  1    GTAB_addPanelPair(REST_TabObject,
2591  1                      (TButPtr)REST_RestoreWin->MarkSummaryTab,
2592  1                      REST_RestoreWin->MarkSummaryPanel);
2593  1    GTAB_addPanelPair(REST_TabObject,
2594  1                      (TButPtr)REST_RestoreWin->MediaTab,
2595  1                      REST_RestoreWin->MediaPanel);
2596  1    GTAB_addPanelPair(REST_TabObject,
2597  1                      (TButPtr)REST_RestoreWin->ViewOptionsTab,
2598  1                      REST_RestoreWin->ViewOptionsPanel);

2600  1    /* Get the icons used */
2601  1    REST_ClientIcon     = GICON_GetIconBySize (
                                              I_GENERICCLIENT, ICON_SMALL);
2602  1    REST_FSWorkItemIcon = GICON_GetIconBySize (I_FILESYSWI, ICON_SMALL);
2603  1    REST_FailedWorkItemIcon = GICON_GetIconBySize (
                                            I_UNUSED_WORKITEM, ICON_SMALL);
2604  1    REST_DirClosedIcon  = GICON_GetIconBySize (I_DIRCLOSED, ICON_SMALL);
2605  1    REST_DirOpenIcon    = GICON_GetIconBySize (I_DIROPEN, ICON_SMALL);
2606  1    REST_FileIcon       = GICON_GetIconBySize (I_FILE, ICON_SMALL);
2607  1    REST_CheckIcon      = GICON_GetIconBySize (I_CHECK, ICON_SMALL);
2608  1    REST_BadIcon        = GICON_GetIcon (I_BADBUOBJECT);
```

```
2609 1    REST_ExpiredIcon     = GICON_GetIconBySize (I_ERROR, ICON_SMALL);
2610 1    REST_MissingChildrenIcon = GICON_GetIconBySize (
                                                    I_WARNING, ICON_SMALL);
2611 1    REST_FailedIcon      = GICON_GetIconBySize (I_ERROR, ICON_SMALL);

2613 1    /* Initialize the other restore pieces */
2614 1    REST_UtilInitialize ();
2615 1    REST_FileInitialize ();
2616 1    REST_SearchInitialize ();
2617 1    REST_SelInitialize ();

2619 1    /* Initialize the sort radio buttons to the initial sort type */
2620 1    sortType = REST_GetSort ();
2621 1    switch (sortType)
2622 2    {
2623 2      case REST_ByName:
2624 2        TBUT_SetSelected ((
                        TButPtr)REST_RestoreWin->NameSortButton, BOOL_TRUE);
2625 2        break;
2626 2      case REST_ByType:
2627 2        TBUT_SetSelected ((
                        TButPtr)REST_RestoreWin->TypeSortButton, BOOL_TRUE);
2628 2        break;
2629 2      case REST_ByDate:
2630 2        TBUT_SetSelected ((
                        TButPtr)REST_RestoreWin->DateSortButton, BOOL_TRUE);
2631 2        break;
2632 2      case REST_BySize:
2633 2        TBUT_SetSelected ((
                        TButPtr)REST_RestoreWin->SizeSortButton, BOOL_TRUE);
2634 2        break;
2635 2      case REST_ByOwner:
2636 2        TBUT_SetSelected ((
                       TButPtr)REST_RestoreWin->OwnerSortButton, BOOL_TRUE);
2637 2        break;
2638 2      default:
2639 2        TBUT_SetSelected ((
                        TButPtr)REST_RestoreWin->NameSortButton, BOOL_TRUE);
2640 2        break;
2641 1    }

2643 1    /* Initialize the options to their initial states */
2644 1    TBUT_SetSelected ((TButPtr)REST_RestoreWin->HiddenButton,
2645 1                      REST_ShowHiddenFiles);
2646 1    TBUT_SetSelected ((TButPtr)REST_RestoreWin->BadFilesButton,
2647 1                      REST_ShowBadFiles);
2648 1    TBUT_SetSelected ((TButPtr)REST_RestoreWin->MarkBadButton,
2649 1                      REST_MarkBadFiles);

2651 1    /* Determine if we want to show the client name */
2652 1    if (REST_RestoreFromClient)
2653 1      hostName = REST_RestoreClient;
2654 1    else
2655 1      hostName = NULL;

2657 1    /* Set the Window and Icon Labels */
2658 1    GUTIL_SetDefaultWindowTitle ((WinPtr)REST_RestoreWin, hostName);
2659 1    WIN_SetIcon ((WinPtr)REST_RestoreWin,
2660 1                 GICON_GetIconBySize (I_RESTORE, ICON_LARGE));

2662 1    GUTIL_CBOX_AllowFreeStyle (
                       REST_RestoreWin->TemplateBox,BACKUP_EXCLUDE_CHARS);
2663    }
```



```
2665    /*****************************************************************
2666     * REST_Display
2667     *
2668     * Description:
2669     *   This routine will display the restore window in an initial state.
2670     *   This can be used the first time the window is displayed or
                                                                  whenever
2671     *   the window needs to be reset to the initial state.
                                                        It assumes the
2672     *   window resources have already been loaded.
                                                    It does not reset user
2673     *   selections for preferences.
2674     *
2675     * Parameters:
2676     *   None.
2677     *
2678     * Returns:
2679     *   None.
2680     *
2681     *****************************************************************/

2683    void REST_Display (void)
2684 1  {
2685 1    eerrno_ty        eerrno;    /* Error Status */

2687 1    /* Reset the current info */
2688 1    currentWorkItemInfo = NULL;

2690 1    /* Initialize this restoral */
2691 1    if ((eerrno = EDMRST_Initialize (GUTIL_GetThisHost (),
2692 1                                     &GREST_Handle,
2693 1                                     CONNECT_TIMEOUT_SEC)) != E_SUCCESS)
2694 2    {

2696 2      /* Couldn't initialize, nothing we can do! */
2697 2      REST_DisplayErrnoMessage ((WinPtr)REST_RestoreWin,
2698 2                                NULL,
2699 2                                REST_GetErrorString(REST_INIT_FAILURE),
2700 2                                eerrno);
2701 2      _exit (-1);
2702 1    }

2704 1    /* Set buttons and labels to default values */
2705 1    TBUT_SetSelected ((
                    TButPtr)REST_RestoreWin->AllowPartialButton, BOOL_TRUE);
2706 1    TED_SetStr ((TEdPtr)REST_RestoreWin->RestoreItemsTarea, "");
2707 1    TED_SetStr ((TEdPtr)REST_RestoreWin->RestoreSizeTarea, "");
2708 1    TED_SetStr ((TEdPtr)REST_RestoreWin->BadFilesText, "");

2710 1    /* Start off with any clients */
2711 1    REST_ShowClients (&currentClientList, currentWIList);

2713 1    /* Initialize the state of the buttons */
2714 1    REST_LBoxSelectionNfy ( NULL );
2715 1    REST_UpdateRemoveButtons ();

2717 1    /* Nothing is selcted */
2718 1    REST_SetSelectionString ("");

2720 1    /* Display the file manager */
2721 1    GFMGR_Display (REST_GetFMgrContext());

2723    }
```

```
2725  /*****************************************************************
2726   * REST_ClearSession
2727   *
2728   * Description:
2729   *   This routine will free up all memory and clear all lists
                                                              assosicated
2730   *   with the current restore session. If the resetFlag is true,
                                                                 then the
2731   *   current marks will be unmarked. If False,
                                            we are exitting so there is
2732   *   no need.
2733   *
2734   * Parameters:
2735   *   resetFlag - Flag if the session is being reset (versus exit)
2736   *
2737   * Returns:
2738   *   None.
2739   *
2740   *****************************************************************/

2742  void REST_ClearSession (BoolEnum  resetFlag)
2743 1 {
2744 1   RestoreInfoPtr  thisInfo;  /* Pointer to walk the list with */
2745 1   RestoreInfoPtr  nextInfo;  /* Next Pointer in the list */

2747 1   /* If this is a reset operation, clear the marks and media */
2748 1   if (resetFlag)
2749 2   {
2750 2     /* Clear out the list boxes */
2751 2     REST_RemoveAllSelectedItems ();
2752 2     REST_RemoveAllMedia ();
2753 1   }

2755 1   /* Clear out the file manager */
2756 1   GFMGR_ClearAll (REST_GetFMgrContext());

2758 1   /* Free up each top level object (
                                 will recursively free up all children) */
2759 1   thisInfo = (RestoreInfoPtr) GFMGR_GetVeryFirstObject (
                                                  REST_GetFMgrContext());
2760 1   while (thisInfo != NULL)
2761 2   {
2762 2     nextInfo = thisInfo->next;
2763 2     REST_FreeInfo (thisInfo);
2764 2     thisInfo = nextInfo;
2765 1   }

2767 1   /* Finalize the Restore process */
2768 1   EDMRST_Finish (GREST_Handle);
2769 1   GREST_Handle = NULL;

2771 1   /* Remove the search window if it is up */
2772 1   REST_SearchRemove ();

2774  }
```



```
2776  /*****************************************************************
2777   * REST_Remove
2778   *
2779   * Description:
2780   *   This routine will remove the restore dialog from the display
                                                                  after
2781   *   verification with the user.
2782   *
2783   * Parameters:
2784   *   None.
2785   *
2786   * Returns:
2787   *   BOOL_TRUE  - If the restore window was really removed
2788   *   BOOL_False - Otherwise
2789   *
2790   *****************************************************************/

2792  BoolEnum REST_Remove (void)
2793 1 {
2794 1   WinPtr    parent;
2795 1   BoolEnum  OKToExit;  /* Flag if user really wants to exit */

2797 1   /* Don't allow exit if we're searching,
                                          or a restore is in progress */
2798 1   if (REST_SearchInProgress() || REST_RestoreInProgress())
2799 1     return (BOOL_FALSE);

2801 1   /* Use the restore window if it is currently visible */
2802 1   if (WIN_IsOpen ((WinPtr)REST_RestoreWin))
2803 1     parent = (WinPtr)REST_RestoreWin;
2804 1   else
2805 1     parent = NULL;

2807 1   /* Verify that the user really wants to end the session */
2808 1   OKToExit =  (GALERT_DisplayQuestion(parent,
2809 1                                       REST_GetErrorString(
                                                  REST_WARNING_INDEX),
2810 1                                       GICON_GetQuestion(),
2811 1                                       REST_GetErrorString (
                                                   REST_IS_OK_TO_END),
2812 1                                       BOOL_FALSE) ==
                                                  GALERT_Affirmative);

2814 1   /* If the user says so, go ahead with the termination */
2815 1   if (OKToExit)
2816 2   {
2817 2     REST_ClearSession (BOOL_FALSE);
2818 1   }

2820 1   return (OKToExit);
2821  }
```

```
2823
2824  1  void REST_SignalHandler (int sig)
2825  1  {
2826  1      /* Clean up help */
2828  1      EDMHELP_End();

2829  1      /* Call the generic signal handler to clean up */
2830      GUTIL_GenericSignalHandler(sig);
   }
```

```
/*****************************************************************************
 *  Restore API Utilities
 *
 *  This file contains utilities to the restore API.  It provides functions
 *  that simply wrap API function calls to provide a more usable interface
 *  for the GUI code layer.
 *
 *  Revision              Date      Author
 *  -----------          --------   ------
 *  Original             06/01/99    JSP
 *
 *****************************************************************************/

#define  ERR_LIB  RESTORE

#include <sys/stat.h>   /* WARNING!!!!! UNIX Dependency!!! */

#include "restAPIUtils.h"


char *        GREST_GetPermissionsString (serverHandle  handle,
                                          GREST_Object  thisObject)
{
  static Char  returnString[PERM_STRING_LENGTH];
  ushort_t     mode;
  int          i;

  mode = EDMRST_GetObjectPermissions(handle, thisObject);

  if (!EDMRST_IsObjectTopLevel(handle, thisObject))
  {
    for (i=0; i<PERM_STRING_LENGTH-1; i++)
    {
      returnString[i] = '-';
    }

    if (S_ISFIFO(mode))
    {
      returnString[0] = 'f';
    }

    else if (S_ISCHR(mode))
    {
      returnString[0] = 'c';
    }

    else if (S_ISDIR(mode))
    {
      returnString[0] = 'd';
    }

    else if (S_ISBLK(mode))
    {
      returnString[0] = 'b';
    }

    else if (S_ISREG(mode))
    {
      returnString[0] = '-';
    }

    else if (S_ISLNK(mode))
```



```
    {
      returnString[0] = 'l';
    }

    else if (S_ISSOCK(mode))
    {
      returnString[0] = 's';
    }

    else
    {
      returnString[0] = '?';
    }

    /*
     * Owner bits
     */

    if (0 != (mode & S_IRUSR))
    {
      returnString[1] = 'r';
    }
    if (0 != (mode & S_IWUSR))
    {
      returnString[2] = 'w';
    }
    if (0 != (mode & S_IXUSR))
    {
      if (0 == (mode & S_ISUID))
      {
        returnString[3] = 'x';
      }
      else
      {
        returnString[3] = 's'; /* setuid */
      }
    }
    else if (0 != (mode & S_ISUID))
    {
      returnString[3] = 'S'; /* silly mode #1 */
    }

    /*
     * Group bits
     */

    if (0 != (mode & S_IRGRP))
    {
        returnString[4] = 'r';
    }
    if (0 != (mode & S_IWGRP))
    {
        returnString[5] = 'w';
    }
    if (0 != (mode & S_IXGRP))
    {
        if (0 == (mode & S_ISGID))
        {
            returnString[6] = 'x';
        }
        else
        {
            returnString[6] = 's';    /* setgid */
        }
    }
    else if (0 != (mode & S_ISGID))
```

```
129  3      {
130  3          returnString[6] = 'S';        /* silly mode #2 */
131  2      }

133  2      /*
134  2       * World bits
135  2       */

137  2      if (0 != (mode & S_IROTH))
138  3      {
139  3          returnString[7] = 'r';
140  2      }
141  2      if (0 != (mode & S_IWOTH))
142  3      {
143  3          returnString[8] = 'w';
144  2      }
145  2      if (0 != (mode & S_IXOTH))
146  3      {
147  3          if (0 == (mode & S_ISVTX))
148  4          {
149  4              returnString[9] = 'x';
150  3          }
151  3          else
152  4          {
153  4              returnString[9] = 't';    /* sticky aka "twiddle" */
154  3          }
155  2      }
156  2      else if (0 != (mode & S_ISVTX))
157  3      {
158  3          returnString[9] = 'T';        /* silly mode #3 */
159  2      }

161  2      returnString[10] = '\0';
162  1    }
163  1    else
164  2    {
165  2      STR_Cpy (returnString, "");
166  1    }

168  1    return (returnString);
169    }
```



```
171    /*******************************************************************
172     * GREST_IsObjectMarkable
173     *
174     * Description:
175     *   This routine will determine if the given object is markable.
176     *
177     * Parameters:
178     *   serverHandle (I) - The handle to the API
179     *   object       (I) - The object to check markability for
180     *
181     * Returns:
182     *   BOOL_TRUE  - If the object can be marked
183     *   BOOL_FALSE - If the object cannot be marked
184     *
185     *******************************************************************/

187    BoolEnum GREST_IsObjectMarkable (serverHandle serverHandle,
188                                     GREST_Object object)
189  1 {
190  1   boolean_ty    isMarkable = BOOL_FALSE;

192  1   if ((serverHandle != NULL) && (object != NULL))
193  2   {
194  2     EDMRST_IsObjectMarkable (serverHandle, object, &isMarkable);
195  1   }

197  1   return (isMarkable);
198    }
```

```
200     BoolEnum GREST_IsObjectMarkableForTime (serverHandle  handle,
201                                             GREST_Object  thisObject,
202                                             time_t        ObjectTime)
203   1 {
204   1   time_t        Pre_mark_time;
205   1   BoolEnum      retValue = BOOL_FALSE;
206   1   u_long        flags;

208   1   if (TBUT_GetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton))
209   1     flags = BACKUP_SELECTION_FLAG_PARTIAL_OK;
210   1   else
211   1     flags = BACKUP_SELECTION_FLAG_COMPLETE_ONLY;

213   1   /*
214   1    *    Get the current backup time,
                                  used to push and pop current backup time.
215   1    */

217   1   if (E_SUCCESS != EDMRST_GetCurrentBackupTime (
                                                    handle, &Pre_mark_time))
218   2   {
219   2     return (BOOL_FALSE);
220   1   }

222   1   /*
223   1    *    Set backup to the desired backup time for marking.
224   1    */

226   1   if (Pre_mark_time != ObjectTime)
227   2   {
228   2     if (E_SUCCESS != EDMRST_SetBackupForTime (
                                                  handle, ObjectTime, flags))
229   3     {
230   3       (void) EDMRST_SetBackupForTime (handle, Pre_mark_time, flags);
231   3       return (BOOL_FALSE);
232   2     }
233   1   }

235   1   /*
236   1    * Check if this object is markable
237   1    */

239   1   EDMRST_IsObjectMarkable (handle, thisObject, &retValue);

241   1   /*
242   1    *    Reset to the pre-mark time.
243   1    */

245   1   if (Pre_mark_time != ObjectTime)
246   2   {
247   2     EDMRST_SetBackupForTime (handle, Pre_mark_time, flags);
248   1   }

250   1   return (retValue);

252     } /* end of GREST_IsObjectMarkableForTime () */
```



```
254     BoolEnum GREST_IsObjectMarkedForTime (serverHandle  handle,
255                                           GREST_Object  thisObject,
256                                           time_t        ObjectTime)
257   1 {
258   1   time_t        Pre_mark_time;
259   1   BoolEnum      retValue = BOOL_FALSE;
260   1   GREST_Object  objectArray[1];
261   1   unsigned long numChecked;
262   1   boolean_ty    markArray[1];
263   1   u_long        flags;

265   1   if (TBUT_GetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton))
266   1     flags = BACKUP_SELECTION_FLAG_PARTIAL_OK;
267   1   else
268   1     flags = BACKUP_SELECTION_FLAG_COMPLETE_ONLY;

270   1   /*
271   1    *    Get the current backup time,
                                  used to push and pop current backup time.
272   1    */

274   1   if (E_SUCCESS != EDMRST_GetCurrentBackupTime (
                                                    handle, &Pre_mark_time))
275   2   {
276   2     return (BOOL_FALSE);
277   1   }

279   1   /*
280   1    *    Set backup to the desired backup time for marking.
281   1    */

283   1   if (Pre_mark_time != ObjectTime)
284   2   {
285   2     if (E_SUCCESS != EDMRST_SetBackupForTime (
                                                  handle, ObjectTime, flags))
286   3     {
287   3       (void) EDMRST_SetBackupForTime (handle, Pre_mark_time, flags);
288   3       return (BOOL_FALSE);
289   2     }
290   1   }

292   1   /*
293   1    * Check if this object is marked
294   1    */

296   1   objectArray[0] = thisObject;
297   1   EDMRST_IsObjectMarked (
                          handle, 1, objectArray, &numChecked, markArray);

299   1   if (numChecked > 0)
300   1     retValue = BOOL_TRUE;

302   1   /*
303   1    *    Reset to the pre-mark time.
304   1    */

306   1   if (Pre_mark_time != ObjectTime)
307   2   {
308   2     EDMRST_SetBackupForTime (handle, Pre_mark_time, flags);
309   1   }

311   1   return (retValue);

313     } /* end of GREST_IsObjectMarkedForTime () */
```

```
/*****************************************************************
 * restCBMgr.c
 *
 *
 * Copyright 1996 by Epoch Systems, Inc.
 *
 *
 * Mission Statement:
 *   This file contains the callback functions necessary for the
 *   EDM Restore window.
 *
 * Required includes:
 *   None
 *
 * Compile-Time Options:
 *   N/A
 *
 *
 * RCS Information:
 *     $RCSfile$
 *     $Revision$
 *     $Date$
 *****************************************************************/

#define  ERR_LIB  RESTORE

/* Muck with the following defines to allow use of putenv which is
                                                          portable */
#define _XOPEN_SOURCE

#include <esl/c_portable.h>
#include <esl/ep_xopen.h>
#include <eerrno/e_errno.h>
#include <util/esl_string.h>

#include <stdlib.h>

#include <libgen.h>
#include <time.h>

#include <apppub.h>
#include <eventpub.h>
#include <rlibpub.h>
#include <gwpub.h>
#include <respub.h>
#include <lboxpub.h>
#include <mbarpub.h>
#include <panelpub.h>
#include <tareapub.h>
#include <tbutpub.h>
#include <tedpub.h>
#include <winpub.h>
#include <strlpub.h>

#include <restore/restore_api.h>
#include "restore.h"
#include "restoreP.h"
#include "restore/restMgr.h"
#include "restCBMgr.h"
#include "restCalendar.h"
#include "restSearch.h"
#include "restSelMgr.h"
#include "restFileMgr.h"
#include "restUtils.h"
#include "gutil/timeUtils.h"
#include "gutil/iconUtils.h"
```



```
#include "gutil/hostUtils.h"
#include "gutil/winUtils.h"
#include "gutil/cboxUtils.h"
#include "gutil/resUtils.h"
#include "gutil/miscUtils.h"
#include "gutil/fileMgr.h"
#include "gutil/tabDefs.h"
#include "gutil/guiDefines.h"
#include "gutil/guiUtils.h"
#include "gutil/alertMgr.h"
#include "help/helpDefs.h"
#include "gutil/fBrowseMgr.h"
#include "gutil/sortUtils.h"
#include "help/helpIpc.h"

ERR_EXTERN
ERR_INMODULE("restore")


/*************
 * Constants *
 *************/

#define TIMEOUT_MINUTES           90
#define TIMEOUT_DELAY             (
                                  TIMEOUT_MINUTES * SECONDS_PER_MINUTE * 1000)
#define TIMEOUT_WARNING_SECONDS   30

#define MAX_INSERT_SORT_ELEMENTS  10

/*************************
 * Local Global Variables *
 *************************/

/* Flag if the CBox was set by code */
static BoolEnum  setBoxByCode = BOOL_FALSE;

/* Current sort type */
static REST_SortType REST_CurrentSort = REST_ByName;

/* Client data for timeout callback */
static Int *timerData = NULL;

/*****************************************************************
 * REST_UpdateDateButtons
 *
 * Description:
 *   This routine will update the backup date buttons sensitivity.
 *   If there are previous backups the previous button will be enabled,
 *   etc...
 *
 * Parameters:
 *   None.
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void REST_UpdateDateButtons (void)
{
  BoolEnum   prevEnabled;
                          /* Flag if previous button should be enabled */
  BoolEnum   nextEnabled;
                              /* Flag if next button should be enabled */
  BoolEnum   calendarEnabled;
```

```
                                /* Flag if calendar button should be enabled */
129 1   boolean_ty status;      /* Return status from API */
130 1   BoolEnum   displayError = BOOL_FALSE; /* Flag if an error occured */
131 1   u_long    flags;

133 1   if (TBUT_GetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton))
134 1     flags = BACKUP_SELECTION_FLAG_PARTIAL_OK;
135 1   else
136 1     flags = BACKUP_SELECTION_FLAG_COMPLETE_ONLY;

138 1   /* If there is a restore in progress, don't update anything */
139 1   if (REST_RestoreInProgress ())
140 1     return;

142 1   /* Check to see if there is a previous backup */
143 1   if (EDMRST_IsTherePrevBackup (
                                GREST_Handle, flags, &status) == E_SUCCESS)
144 2   {
145 2     if (status)
146 3     {
147 3       prevEnabled = BOOL_TRUE;
148 2     }
149 2     else
150 3     {
151 3       prevEnabled = BOOL_FALSE;
152 2     }
153 1   }
154 1   else
155 2   {
156 2     displayError = BOOL_TRUE;
157 2     prevEnabled = BOOL_TRUE;
158 1   }

160 1   /* Check to see if there is a next backup */
161 1   if (EDMRST_IsThereNextBackup (
                                  GREST_Handle, 0, &status) == E_SUCCESS)
162 2   {
163 2     if (status)
164 3     {
165 3       nextEnabled = BOOL_TRUE;
166 2     }
167 2     else
168 3     {
169 3       nextEnabled = BOOL_FALSE;
170 2     }
171 1   }
172 1   else
173 2   {
174 2     displayError = BOOL_TRUE;
175 2     nextEnabled = BOOL_TRUE;
176 1   }

178 1   /* If there is either a prev or next backup, enable the calendar */
179 1   calendarEnabled = (prevEnabled || nextEnabled);

181 1   /* Set the state of the previous button as determined */
182 1   GUTIL_WGT_SetEnabled ((
                  WgtPtr)REST_RestoreWin->PrevBackupButton, prevEnabled);

184 1   /* Set the state of the next button as determined */
185 1   GUTIL_WGT_SetEnabled ((
                  WgtPtr)REST_RestoreWin->NextBackupButton, nextEnabled);

187 1   /* Set the state of the calendar button as determined */
188 1   GUTIL_WGT_SetEnabled ((
                WgtPtr)REST_RestoreWin->CalendarButton, calendarEnabled);
```



```
190 1   if (displayError && !REST_IsReReadInProgress ())
191 2   {
192 2     GALERT_DisplayError ((WinPtr)REST_RestoreWin,
193 2                          REST_GetErrorString (REST_WARNING_INDEX),
194 2                          GICON_GetWarning(),
195 2                          REST_GetErrorString (
                                        REST_IS_PREV_NEXT_BACKUP_ERROR));
196 1   }
197     }
```

```
199  /*****************************************************************
200   * REST_SelectCurrentTemplate
201   *
202   * Description:
203   *   This routine will select the current template in the template
204   *   box.
205   *
206   * Parameters:
207   *   None.
208   *
209   * Returns:
210   *   None.
211   *
212   *****************************************************************/

214  void  REST_SelectCurrentTemplate (void)
215 1{
216 1  setBoxByCode = BOOL_TRUE;
217 1  CBOX_CurSelect (REST_RestoreWin->TemplateBox);
218 1  setBoxByCode = BOOL_FALSE;
219  }
```



```
221  /*****************************************************************
222   * REST_SelectCurrentTrail
223   *
224   * Description:
225   *   This routine will select the current trail in the primary
226   *   box.
227   *
228   * Parameters:
229   *   None.
230   *
231   * Returns:
232   *   None.
233   *
234   *****************************************************************/

236  void  REST_SelectCurrentTrail (void)
237 1{
238 1  setBoxByCode = BOOL_TRUE;
239 1  CBOX_CurSelect (REST_RestoreWin->PrimaryBox);
240 1  setBoxByCode = BOOL_FALSE;
241  }
```

```
243  /*****************************************************************
244   * REST_SetFSOptionsVisibility
245   *
246   * Description:
247   *   This routine will set the visibility status of the File System
248   *   Backup options panel and its widgets.  If the visibility passed
249   *   is BOOL_TRUE it will enable all the widgets, if BOOL_FALSE it
250   *   will disable and clear all the widgets.
251   *
252   * Parameters:
253   *   visible (I) - Value of the visibility to set
254   *
255   * Returns:
256   *   None.
257   *
258   *****************************************************************/

260  void  REST_SetFSOptionsVisibility (BoolEnum visible)
261 1{

263 1   /* If there is a restore in progress, don't update anything */
264 1   if (REST_RestoreInProgress ())
265 1     return;

267 1   /* Set the enabling of the entire panel based on the given
                                                          visibility */
268 1   if (WGT_IsEnabled ((
                      WgtPtr)REST_RestoreWin->FSOptionsPanel) != visible)
269 1     GUTIL_WGT_SetEnabled ((
                       WgtPtr)REST_RestoreWin->FSOptionsPanel, visible);

271 1   /* If the visibility is false, clear the widgets */
272 1   if (!visible)
273 2   {

275 2     /* Clear out any old Templates: */
276 2     CBOX_GoFirst(REST_RestoreWin->TemplateBox);
277 2     while (CBOX_IsOk (REST_RestoreWin->TemplateBox))
278 3     {
279 3       CBOX_GoFirst (REST_RestoreWin->TemplateBox);
280 3       CBOX_CurRemoveElt (REST_RestoreWin->TemplateBox);
281 2     }

283 2     /* Clear out the old trails: */
284 2     CBOX_GoFirst(REST_RestoreWin->PrimaryBox);
285 2     while (CBOX_IsOk (REST_RestoreWin->PrimaryBox))
286 3     {
287 3       CBOX_GoFirst (REST_RestoreWin->PrimaryBox);
288 3       CBOX_CurRemoveElt (REST_RestoreWin->PrimaryBox);
289 2     }

291 2     /* Clear the backup date */
292 2     TED_SetStr ((TEdPtr)REST_RestoreWin->BackupDateText, "");

294 1   }
295  }
```



```
297  /*****************************************************************
298   * REST_SetTemplateBoxes
299   *
300   * Description:
301   *   This routine will set the template and trail combo boxes to
302   *   the given values.
303   *
304   * Parameters:
305   *   None.
306   *
307   * Returns:
308   *   None.
309   *
310   *****************************************************************/

312  void REST_SetTemplateBoxes (Str      template,
313                              BoolEnum isAlternate)
314 1{
315 1   BoolEnum found = BOOL_FALSE; /* Flag for finding the template */

317 1   /* Find the current template in the template box and select it */
318 1   CBOX_GoFirst(REST_RestoreWin->TemplateBox);
319 1   while ((!found) && (CBOX_IsOk (REST_RestoreWin->TemplateBox)))
320 2   {

322 2     /* Use the label to determine if this is the current template */
323 2     if (STR_Cmp(template, CBOX_CurGetLabel(
                                    REST_RestoreWin->TemplateBox)) == 0)
324 3     {
325 3       /* This be the one, select it */
326 3       REST_SelectCurrentTemplate ();
327 3       found = BOOL_TRUE;
328 2     }
329 2     else
330 3     {
331 3       /* Not this one, try the next one */
332 3       CBOX_GoNext(REST_RestoreWin->TemplateBox);
333 2     }
334 1   }

336 1   /* Select the first for primary or the second for the alternate */
337 1   CBOX_GoFirst(REST_RestoreWin->PrimaryBox);
338 1   if (isAlternate)
339 1     CBOX_GoNext (REST_RestoreWin->PrimaryBox);
340 1   REST_SelectCurrentTrail ();
341  }
```

```
343  /*****************************************************************
344   * REST_UpdateTemplateFromBoxes
345   *
346   * Description:
347   *   This routine will update the current template based on the
348   *   user selection of a new template/trail from the combo boxes.
349   *   It will do nothing if this routine was called as a result of
350   *   a function call to select a CBox item (as long as the global
351   *   setBoxByCode variable was set to true).
352   *
353   * Parameters:
354   *   None.
355   *
356   * Returns:
357   *   None.
358   *
359   *****************************************************************/

361  void  REST_UpdateTemplateFromBoxes (void)
362  1 {
363  1   GREST_TemplateName currentTemplate;      /* Current template name */
364  1   BoolEnum           currentAlternate;     /* Current trail */
365  1   GREST_TemplateName template;             /* Template to set to */
366  1   BoolEnum           alternate;            /* Trail to set to */
367  1   eerrno_ty          eerrno;               /* Error status */
368  1   GREST_Object       objects[1];
                                        /* Bogus buffer to get objects */
369  1   long               cookie = INIT_COOKIE; /* Ah, the magic cookie */
370  1   long               numEntries;
                                      /* Bogus value for object count */
371  1   BoolEnum           removeSearch = BOOL_FALSE;
                                         /* Should search be removed */

373  1   /* Do nothing if this selection is due to code (
                                                   not user selection) */
374  1   if (setBoxByCode)
375  1     return;

377  1   /* Process events to update the button while the user waits */
378  1   EVENT_ProcessPending ();

380  1   /* Get the current template we are using */
381  1   eerrno = EDMRST_GetCurrentTemplate (GREST_Handle,
382  1                                       currentTemplate,
383  1                                       &currentAlternate);

385  1   /* Get the selected template name */
386  1   if (CBOX_ChosenGetLabel (REST_RestoreWin->TemplateBox) != NULL)
387  2   {
388  2     STR_Cpy (template, CBOX_ChosenGetLabel (
                                        REST_RestoreWin->TemplateBox));
389  1   }
390  1   else if (errno == E_SUCCESS)
391  2   {
392  2     STR_Cpy (template, currentTemplate);
393  1   }
394  1   else
395  2   {
396  2     STR_Cpy (template, "");
397  1   }

399  1   /* Get the current value for the primary/alternate box */
400  1   if (CBOX_ChosenGetId (REST_RestoreWin->PrimaryBox) == 1)
401  1     alternate = BOOL_FALSE;
402  1   else
```



```
403  1     alternate = BOOL_TRUE;

405  1   /* If we have no current template or the template has changed. */
406  1   if ((eerrno != E_SUCCESS) ||
407  1       (STR_Cmp (template, currentTemplate) != CMP_EQUAL) ||
408  1       (alternate != currentAlternate))
409  2   {

411  2     /* If the search window is displayed, ask user first */
412  2     if (REST_SearchWindowIsDisplayed ())
413  3     {

415  3       if (GALERT_DisplayQuestion ((WinPtr)REST_RestoreWin,
416  3                                   REST_GetErrorString (
                                                 REST_WARNING_INDEX),
417  3                                   GICON_GetWarning(),
418  3                                   REST_GetErrorString (
                                          REST_TEMP_SWITCH_SEARCH_WARNING),
419  3                                   BOOL_FALSE) == GALERT_Affirmative)
420  4       {
421  4         removeSearch = BOOL_TRUE;
422  3       }
423  3       else
424  4       {
425  4         /* Set back to the original */
426  4         REST_SetTemplateBoxes (currentTemplate, currentAlternate);

428  4         /* Get out */
429  4         return;
430  3       }
431  2     }

433  2     /* If anything is marked, verify with user and unmark */
434  2     LBOX_GoHome (REST_RestoreWin->SelectedListBox);
435  2     if (LBOX_CurGetClientData(
                              REST_RestoreWin->SelectedListBox) != NULL)
436  3     {

438  3       /* We have marked items, ask user before switching */
439  3       if (GALERT_DisplayQuestion ((WinPtr)REST_RestoreWin,
440  3                                   REST_GetErrorString(
                                                 REST_WARNING_INDEX),
441  3                                   GICON_GetWarning(),
442  3                                   REST_GetErrorString(
                                            REST_TEMP_SWITCH_WARNING),
443  3                                   BOOL_FALSE) != GALERT_Affirmative)
444  4       {
445  4         /* Set back to the original */
446  4         REST_SetTemplateBoxes (currentTemplate, currentAlternate);

448  4         /* Get out */
449  4         return;
450  3       }
451  2     }

453  2     /* Set the current template */
454  2     if ((currentWorkItemInfo != NULL) &&
455  2         (currentWorkItemInfo->restoreObject != NULL))
456  3     {
457  3       if((eerrno = EDMRST_SetTopLevelTemplate(GREST_Handle,
458  3
                                     currentWorkItemInfo->restoreObject,
459  3                                            template,
460  3                                            alternate)) == 0)
461  4       {
```

```
463 4           /* Remove the search dialog if necessary */
464 4           if (removeSearch)
465 4             REST_SearchRemove ();

467 4           /* Create one object */
468 4           EDMRST_AllocRestorableObjects (GREST_Handle, objects, 1);

470 4           /* Attempt to get the object */
471 4           if ((eerrno = EDMRST_GetRestorableObjects(GREST_Handle,
472 4
                                        currentWorkItemInfo->restoreObject,
473 4                                                 BOOL_TRUE,
474 4                                                 1,
475 4                                                 objects,
476 4                                                 &numEntries,
477 4                                                 &cookie)) == 0)
478 5           {

480 5             /* Update the date (
                                  this will re-read the work-item for us) */
481 5             REST_UpdateBackupDate ();

483 5             /* Clear out any selection data */
484 5             REST_ClearObjectMarks (currentWorkItemInfo);
485 4           }
486 4           else
487 5           {
488 5             /* Go back to the original template/trail */
489 5             EDMRST_SetTopLevelTemplate (GREST_Handle,
490 5
                                        currentWorkItemInfo->restoreObject,
491 5                                        currentTemplate,
492 5                                        currentAlternate);

494 5             /* Get one object to init the work-item */
495 5             EDMRST_GetRestorableObjects (GREST_Handle,
496 5
                                        currentWorkItemInfo->restoreObject,
497 5                                         BOOL_TRUE,
498 5                                         1,
499 5                                         objects,
500 5                                         &numEntries,
501 5                                         &cookie);
502 4           }
503 3         }

505 3         /* Free up the object */
506 3         EDMRST_FreeRestorableObjects (GREST_Handle, objects, 1);

508 3         if (eerrno != 0)
509 4         {
510 4           Char outputString[2 * GMAX_OBJECT_LENGTH];
                                          /* Error string to display */

512 4           /* All this, and we couldn't switch, what a shame */
513 4           if (alternate)
514 5           {
515 5             STR_Sprintf (outputString,
516 5                          REST_GetErrorString (REST_TEMP_SWITCH_ERROR),
517 5                          (Str)STRL_GetNthStr(REST_TrailNameList,
518 5                                              REST_ALTERNATE_TRAIL_INDEX),
519 5                          template);
520 4           }
521 4           else
522 5           {
523 5             STR_Sprintf (outputString,
```



```
524 5                          REST_GetErrorString (REST_TEMP_SWITCH_ERROR),
525 5                          (Str)STRL_GetNthStr(REST_TrailNameList,
526 5                                              REST_PRIMARY_TRAIL_INDEX),
527 5                          template);
528 4           }
529 4           REST_DisplayErrnoMessage ((WinPtr)REST_RestoreWin,
530 4                                     NULL,
531 4                                     outputString,
532 4                                     eerrno);

534 4           /*
535 4            * Unfortunately, it is too late to keep the current marks (
                                                                    if there
536 4            * were any).  I know,
                                  big suck.  At least show that there are no
537 4            * longer an marked objects to the user.
538 4            */

540 4           REST_ClearObjectMarks (currentWorkItemInfo);
541 4           GFMGR_Display (REST_GetFMgrContext());

543 4           /* Set back to the current template and trail */
544 4           REST_SetTemplateBoxes (currentTemplate, currentAlternate);

546 3         }
547 2       }
548 1     }

550     }
```

```
552  /*****************************************************************
553   * REST_ClearRestoreObjects
554   *
555   * Description:
556   *   This routine will free and reset the restore object of the
                                                                  children
557   *   of the given parent
558   *
559   * Parameters:
560   *   parent (I) - The parent to start with
561   *
562   * Returns:
563   *   None.
564   *
565   *****************************************************************/

567  static void REST_ClearRestoreObjects (RestoreInfoPtr parent)
568 1 {
569 1   RestoreInfoPtr nextChild;

571 1   if (parent != NULL)
572 2   {
573 2     nextChild = parent->children;
574 2     while (nextChild != NULL)
575 3     {
576 3       REST_ClearRestoreObjects (nextChild);
577 3       EDMRST_FreeRestorableObjects (
                          GREST_Handle, &nextChild->restoreObject, 1);
578 3       nextChild->restoreObject = NULL;

580 3       nextChild = nextChild->next;
581 2     }
582 1   }
583  }
```



```
585  /*****************************************************************
586   * REST_UpdateBackupOptions
587   *
588   * Description:
589   *   This routine will update the available options visible to the
                                                                  user.
590   *
591   * Parameters:
592   *   info (I) - Currently selected info record
593   *
594   * Returns:
595   *   None.
596   *
597   *****************************************************************/

599  void  REST_UpdateBackupOptions (RestoreInfoPtr info)
600 1 {
601 1   RestoreInfoPtr      tmpObject;
                                  /* Temp record to walk list with */
602 1   RestoreInfoPtr      originalWI;
                                    /* Safe place for original workitem */
603 1   BoolEnum            found = BOOL_FALSE; /* Flag if we found it */
604 1   BoolEnum            newWI = BOOL_FALSE;
                                    /* Flag if the workitem changed */

606 1   /* If this is the top level, disable FS specifics,
                                                        enable all tabs */
607 1   if ((info == NULL) ||
608 1       ((info->type == REST_Client) && (
                                  info != REST_GetCurrentClientInfo ())))
609 2   {
610 2     /* Disable the path widget */
611 2     GUTIL_WGT_SetEnabled ((
                          WgtPtr)REST_RestoreWin->PathTArea, BOOL_FALSE);
612 2     GUTIL_WGT_SetEnabled ((
                          WgtPtr)REST_RestoreWin->PathTEd,   BOOL_FALSE);

614 2     /* Hide the File systems options panel */
615 2     REST_SetFSOptionsVisibility (BOOL_FALSE);
616 2     REST_SetSearchVisiblity (BOOL_FALSE);

618 2     /* Update the partial backup button availability */
619 2     REST_UpdatePartialButton ();

621 2     /* We ain't working with work-items no more */
622 2     if (currentWorkItemInfo != NULL)
623 3     {
624 3       REST_RemoveAllSelectedItems ();
625 3       REST_ClearObjectMarks (currentWorkItemInfo);
626 3       REST_ClearRestoreObjects (currentWorkItemInfo);
627 3       currentWorkItemInfo = NULL;
628 2     }
629 1   }

631 1   else if (info->type == REST_FailedWorkItem)
632 2   {

634 2     /* Disable the path widget */
635 2     GUTIL_WGT_SetEnabled ((
                          WgtPtr)REST_RestoreWin->PathTArea, BOOL_FALSE);
636 2     GUTIL_WGT_SetEnabled ((
                          WgtPtr)REST_RestoreWin->PathTEd,   BOOL_FALSE);

638 2     /* Hide the File systems options panel */
639 2     REST_SetFSOptionsVisibility (BOOL_FALSE);
```

```
640 2     REST_SetSearchVisiblity (BOOL_FALSE);

642 2     /* Update the partial backup button availability */
643 2     REST_UpdatePartialButton ();

645 2     /* We ain't working with a valid workitem any more */
646 2     if (currentWorkItemInfo != NULL)
647 3     {
648 3       REST_ClearObjectMarks (currentWorkItemInfo);
649 3       REST_ClearRestoreObjects (currentWorkItemInfo);
650 3       currentWorkItemInfo = NULL;
651 2     }

653 1   }

655 1   /* Else if this is not the current client,
                                          check the work item status */
656 1   else if (info->type != REST_Client)
657 2   {

659 2     /* Enable the path widget */
660 2     GUTIL_WGT_SetEnabled ((
                          WgtPtr)REST_RestoreWin->PathTArea, BOOL_TRUE);
661 2     GUTIL_WGT_SetEnabled ((
                          WgtPtr)REST_RestoreWin->PathTEd,   BOOL_TRUE);

663 2     /* Keep a pointer to the original work item */
664 2     originalWI = currentWorkItemInfo;

666 2     /*
667 2      * Update FS Options
668 2      */

670 2     /* Find the work-item for this object */
671 2     tmpObject = info;
672 2     while ((!found) && (tmpObject != NULL))
673 3     {
674 3       if (tmpObject->type == REST_WorkItem)
675 4       {
676 4         if (currentWorkItemInfo != tmpObject)
677 5         {
678 5           currentWorkItemInfo = tmpObject;
679 5           newWI = BOOL_TRUE;
680 4         }
681 4         found = BOOL_TRUE;
682 3       }
683 3       else
684 4       {
685 4         tmpObject = tmpObject->parent;
686 3       }
687 2     }

689 2     /*
690 2      * If there are no children for this workitem yet, get them
691 2      */
692 2     if (currentWorkItemInfo->children == NULL)
693 3     {
694 3       REST_CreateInfoChildren (currentWorkItemInfo);
695 2     }


698 2     /*
699 2      * If the workitem is a failed workitem,
                                        unset the current workitem
700 2      */
701 2     if (currentWorkItemInfo->type == REST_FailedWorkItem)
```



```
702 3     {
703 3       currentWorkItemInfo = NULL;
704 2     }

706 2     /* Show the File System work-item options panel if the WI is valid
                                                                        */
707 2     if((currentWorkItemInfo != NULL) && (
                                    currentWorkItemInfo->children != NULL))
708 3     {
709 3       REST_SetFSOptionsVisibility (BOOL_TRUE);
710 3       REST_SetSearchVisiblity (BOOL_TRUE);
711 2     }
712 2     else
713 3     {
714 3       REST_SetFSOptionsVisibility (BOOL_FALSE);
715 3       REST_SetSearchVisiblity (BOOL_FALSE);
716 2     }

718 2     /* Update the partial backup button availability */
719 2     REST_UpdatePartialButton ();

721 2     /*
722 2      * If the user has changed workitems we need to re-read the
                                                                workitem
723 2      * the user is now looking at since the old restorable objects are
724 2      * no good.
725 2      */

727 2     if (newWI)
728 3     {

730 3       /* Clean any existing data */
731 3       if (originalWI != NULL)
732 4       {
733 4         /* Clear out any selection data */
734 4         REST_ClearObjectMarks (originalWI);
735 4         REST_ClearRestoreObjects (originalWI);
736 3       }

738 3       /* If the workitem is valid update the options */
739 3       if (currentWorkItemInfo != NULL)
740 4       {
741 4         /* Update the date (this will re-read the work-item for us) */
742 4         REST_UpdateBackupDate ();

744 4         /* Display the new work-item data */
745 4         REST_UpdateBackupTemplates (currentWorkItemInfo);
746 3       }
747 2     }
748 1   }

750   }
```

```
752  /*****************************************************************
753   * REST_OKToSwitchTimes
754   *
755   * Description:
756   *   This routine determines if it is OK to switch backup times.
757   *
758   * Parameters:
759   *   None.
760   *
761   * Returns:
762   *   None.
763   *
764   *****************************************************************/

766  static BoolEnum REST_OKToSwitchTimes (void)
767 1 {
768 1   boolean_ty   okToSwitch = FALSE;
769 1   BoolEnum     retVal = BOOL_TRUE;

771 1   /*
772 1    * See if it is OK to switch times and leave marks
773 1    */

775 1   if ((currentWorkItemInfo != NULL) &&
776 1       (EDMRST_GetBackupTimesSupport (GREST_Handle,
777 1                                      currentWorkItemInfo->restoreObject,
778 1                                        &okToSwitch) == E_SUCCESS) &&
779 1       (okToSwitch != TRUE))
780 2   {
781 2     /* Check if there are any marks */
782 2     LBOX_GoHome (REST_RestoreWin->SelectedListBox);
783 2     if (LBOX_CurGetClientData(
                              REST_RestoreWin->SelectedListBox) != NULL)
784 3     {
785 3       if (GALERT_DisplayQuestion((WinPtr)REST_RestoreWin,
786 3                                  REST_GetErrorString (
                                               REST_WARNING_INDEX),
787 3                                  GICON_GetWarning(),
788 3                                  REST_GetErrorString(
                                           REST_SWITCH_TIMES_WARNING),
789 3                                  BOOL_FALSE) == GALERT_Affirmative)
790 4       {
791 4         /* Clear out any selection data */
792 4         REST_ClearObjectMarks (currentWorkItemInfo);
793 4         retVal = BOOL_TRUE;
794 3       }
795 3       else
796 4       {
797 4         retVal = BOOL_FALSE;
798 3       }
799 2     }
800 1   }

802 1   return (retVal);
803  }
```



```
805  /*****************************************************************
806   * REST_PrevButtonSelect
807   *
808   * Description:
809   *   This routine handles the user selection of the previous backup
                                                              button.
810   *
811   * Parameters:
812   *   None.
813   *
814   * Returns:
815   *   None.
816   *
817   *****************************************************************/

819  void  REST_PrevButtonSelect (void)
820 1 {
821 1   eerrno_ty    eerrno;   /* Error status */
822 1   u_long       flags;

824 1   /* Verify it is OK to switch times */
825 1   if (!REST_OKToSwitchTimes ())
826 1     return;

828 1   if (TBUT_GetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton))
829 1     flags = BACKUP_SELECTION_FLAG_PARTIAL_OK;
830 1   else
831 1     flags = BACKUP_SELECTION_FLAG_COMPLETE_ONLY;

833 1   /* Try to get a previous backup */
834 1   if ((eerrno = EDMRST_SetPrevBackup (GREST_Handle, flags)) == 0)
835 2   {

837 2     /* Got the previous backup, update the date */
838 2     REST_UpdateBackupDate ();
839 1   }
840 1   else
841 2   {
842 2     REST_DisplayErrnoMessage ((WinPtr)REST_RestoreWin,
843 2                               NULL,
844 2                               REST_GetErrorString (
                                            REST_PREV_BACKUP_ERROR),
845 2                               eerrno);
846 1   }
847  }
```

```
849  /*********************************************************************
850   * REST_NextButtonSelect
851   *
852   * Description:
853   *   This routine handles the user selection of the next backup button.
854   *
855   * Parameters:
856   *   None.
857   *
858   * Returns:
859   *   None.
860   *
861   *********************************************************************/

863  void  REST_NextButtonSelect (void)
864 1{
865 1  eerrno_ty    eerrno;   /* Error status */
866 1  u_long       flags;

868 1  /* Verify it is OK to switch times */
869 1  if (!REST_OKToSwitchTimes ())
870 1    return;

872 1  if (TBUT_GetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton))
873 1    flags = BACKUP_SELECTION_FLAG_PARTIAL_OK;
874 1  else
875 1    flags = BACKUP_SELECTION_FLAG_COMPLETE_ONLY;

877 1  /* Try to get a next backup */
878 1  if ((eerrno = EDMRST_SetNextBackup (GREST_Handle, flags)) == 0)
879 2  {

881 2    /* Got the previous backup, update the date */
882 2    REST_UpdateBackupDate ();
883 1  }
884 1  else
885 2  {
886 2    REST_DisplayErrnoMessage ((WinPtr)REST_RestoreWin,
887 2                              NULL,
888 2                              REST_GetErrorString (
                                                REST_NEXT_BACKUP_ERROR),
889 2                              eerrno);
890 1  }
891  }
```



```
893  /*********************************************************************
894   * REST_CalendarButtonSelect
895   *
896   * Description:
897   *   This routine handles the user selection of the backup calendar
                                                                    button.
898   *   It will display the restoral calendar,
                                                  showing the available backup
899   *   date/times.
900   *
901   * Parameters:
902   *   None.
903   *
904   * Returns:
905   *   None.
906   *
907   *********************************************************************/

909  void  REST_CalendarButtonSelect (void)
910 1{
911 1  time_t       oldTime;  /* The current backup time */
912 1  time_t       newTime;  /* The user selected backup time */
913 1  eerrno_ty    eerrno;   /* Error status */
914 1  u_long       flags;

916 1  if (TBUT_GetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton))
917 1    flags = BACKUP_SELECTION_FLAG_PARTIAL_OK;
918 1  else
919 1    flags = BACKUP_SELECTION_FLAG_COMPLETE_ONLY;

921 1  /* Get the current backup time */
922 1  EDMRST_GetCurrentBackupTime (GREST_Handle, &oldTime);

924 1  /* Get the user selected backup time */
925 1  newTime = REST_GetUserSelectedTime (
                                      currentWorkItemInfo->restoreObject,
926 1                                    oldTime,
927 1                                    (WinPtr)REST_RestoreWin);

929 1  /* If the user picked a different time */
930 1  if ((newTime != 0) && (oldTime != newTime))
931 2  {

933 2    /* Verify it is OK to switch times */
934 2    if (!REST_OKToSwitchTimes ())
935 2      return;

937 2    /* Get the new backup */
938 2    if ((eerrno = EDMRST_SetBackupForTime (
                                      GREST_Handle, newTime, flags)) == 0)
939 3    {

941 3      /* Update the backup date */
942 3      if (currentWorkItemInfo != NULL)
943 4      {
944 4        REST_UpdateBackupDate ();
945 3      }
946 2    }
947 2    else
948 3    {
949 3      Char outputString[2 * GMAX_OBJECT_LENGTH];  /* Error string */

951 3      /* Unable to set to the backup selected */
952 3      STR_Sprintf (outputString,
953 3                   REST_GetErrorString (
```

```
                                        REST_BACKUP_TIME_SWITCH_ERROR),
954 3                   REST_GetTimeDateString (newTime));
955 3       REST_DisplayErrnoMessage ((WinPtr)REST_RestoreWin,
956 3                                 NULL,
957 3                                 outputString,
958 3                                 eerrno);
959 2     }
960 1   }
961     }
```



```
963   /*****************************************************************
964    * REST_CompareProc
965    *
966    * Description:
967    *   This routine is provided to the generic sort routine to compare
968    *   two items.
969    *
970    * Parameters:
971    *   item1 (I) - The first item to compare
972    *   item2 (I) - The second item to compare
973    *
974    * Returns:
975    *   None.
976    *
977    *****************************************************************/

979   static CmpEnum REST_CompareProc (void *item1,
980                                    void *item2)
981 1 {
982 1   RestoreInfoPtr info1;
983 1   RestoreInfoPtr info2;
984 1   CmpEnum        retVal;

986 1   info1 = (RestoreInfoPtr)item1;
987 1   info2 = (RestoreInfoPtr)item2;

989 1   if (REST_IsLessThan (info1, info2))
990 1     retVal = CMP_UNDER;
991 1   else
992 1     retVal = CMP_OVER;

994 1   return (retVal);
995   }
```

```
997  /*****************************************************************
998   * REST_GetNextProc
999   *
1000  * Description:
1001  *   This routine is provided to the generic sort routine to get
1002  *   the next restore info pointer from the given item.
1003  *
1004  * Parameters:
1005  *   item (I) - The item to get the next item from
1006  *
1007  * Returns:
1008  *   the next item in the list.
1009  *
1010  *****************************************************************/

1012  static void *REST_GetNextProc (void *item)
1013 1 {
1014 1   RestoreInfoPtr info;
1015 1   RestoreInfoPtr nextInfo;

1017 1   if (item != NULL)
1018 2   {
1019 2     info = (RestoreInfoPtr)item;
1020 2     nextInfo = info->next;
1021 1   }
1022 1   else
1023 2   {
1024 2     nextInfo = NULL;
1025 1   }

1027 1   return ((void *)nextInfo);
1028  }
```



```
1030  /*****************************************************************
1031   * REST_GetNextAddressProc
1032   *
1033   * Description:
1034   *   This routine is provided to the generic sort routine to get
1035   *   the address of the next field from the given item.
1036   *
1037   * Parameters:
1038   *   item (I) - The item to get the address of the next field from
1039   *
1040   * Returns:
1041   *   the address of the next field for the given item.
1042   *
1043   *****************************************************************/

1045  static void *REST_GetNextAddressProc (void *item)
1046 1 {
1047 1   RestoreInfoPtr info;
1048 1   void           *retAddress;

1050 1   if (item != NULL)
1051 2   {
1052 2     info = (RestoreInfoPtr)item;
1053 2     retAddress = &(info->next);
1054 1   }
1055 1   else
1056 2   {
1057 2     retAddress = NULL;
1058 1   }

1060 1   return (retAddress);
1061  }
```

```
1063  /*****************************************************************
1064   * REST_SetNextProc
1065   *
1066   * Description:
1067   *   This routine is provided to the generic sort routine to set
1068   *   the next restore info pointer for the given item to the given
                                                                  next item.
1069   *
1070   * Parameters:
1071   *   item     (I) - The item to set the next item for
1072   *   nextItem (I) - The next item to set to
1073   *
1074   * Returns:
1075   *   None.
1076   *
1077   *****************************************************************/

1079  static void REST_SetNextProc (void *item,
1080                                void *nextItem)
1081  1 {
1082  1   RestoreInfoPtr info;

1084  1   if (item != NULL)
1085  2   {
1086  2     info = (RestoreInfoPtr)item;
1087  2     info->next = (RestoreInfoPtr)nextItem;
1088  1   }
1089    }
```



```
1091  /*****************************************************************
1092   * REST_SortChildren
1093   *
1094   * Description:
1095   *   This routine will sort the children of the given parent object.
1096   *
1097   * Parameters:
1098   *   parent (I) - The parent object to sort the children for.
1099   *
1100   * Returns:
1101   *   None.
1102   *
1103   *****************************************************************/

1105  void REST_SortChildren (RestoreInfoPtr parent)
1106  1 {
1107  1   RestoreInfoPtr tmpInfo;          /* Pointer to walk the list with */
1108  1   Int            count = 0;
                                          /* Number of children in the list */

1110  1   /* Start at the first child of the parent object */
1111  1   tmpInfo = parent->children;
1112  1   while (tmpInfo != NULL)
1113  2   {
1114  2     count++;
1115  2     tmpInfo = tmpInfo->next;
1116  1   }

1118  1   GSORT_QuickSort ((void **)(&parent->children),
1119  1                    NULL,
1120  1                    count,
1121  1                    REST_CompareProc,
1122  1                    REST_GetNextProc,
1123  1                    REST_GetNextAddressProc,
1124  1                    REST_SetNextProc);

1126  1   /* Loop through the children sorting each child's children */
1127  1   tmpInfo = parent->children;
1128  1   while (tmpInfo != NULL)
1129  2   {
1130  2     /* Sort the children first */
1131  2     REST_SortChildren (tmpInfo);

1133  2     /* Move to the next child */
1134  2     tmpInfo = tmpInfo->next;
1135  1   }

1137    }
```

```
1139  /*****************************************************************
1140   * REST_SetSort
1141   *
1142   * Description:
1143   *   This routine will set the current sort and update the order of
1144   *   the children currently displayed.
1145   *
1146   * Parameters:
1147   *   sortType (I) - The type of sort to perform.
1148   *
1149   * Returns:
1150   *   None.
1151   *
1152   *****************************************************************/

1154  void      REST_SetSort (REST_SortType sortType)
1155  1 {
1156  1   RestoreInfoPtr  info;
                            /* Pointer to walk the object list with */
1157  1   RestoreInfoPtr  foundInfo;
                          /* Object found which is currently selected */

1159  1   /* Freeze the file manager display to avoid major flickering */
1160  1   GFMGR_FreezeDisplay (REST_GetFMgrContext());

1162  1   /* Flag that we are sorting */
1163  1   REST_SetReReadInProgress (BOOL_TRUE);

1165  1   /* Set the current sort type */
1166  1   REST_CurrentSort = sortType;

1168  1  /* Walk all the objects and sort the children */
1169  1   info = (RestoreInfoPtr) GFMGR_GetVeryFirstObject (
                                                  REST_GetFMgrContext());

1170  1   while (info != NULL)
1171  2   {

1173  2     /* Sort the children of this object */
1174  2     REST_SortChildren (info);

1176  2     /* Tell the file manager that the children have been updated */
1177  2     GFMGR_UpdateChildren (REST_GetFMgrContext(), info);

1179  2     /* Move to the next object */
1180  2     info = info->next;
1181  1   }

1183  1   /* Select the object which should be currently selected */
1184  1   foundInfo = REST_FindSelectionString ();
1185  1   if (foundInfo != NULL)
1186  2   {
1187  2     GFMGR_SelectObject (REST_GetFMgrContext(),
1188  2                         (GFMGR_Object)foundInfo,
1189  2                         BOOL_FALSE);
1190  2     REST_SetSelectionString (REST_GetFullName(foundInfo));
1191  1   }

1193  1   /* Update the file manager's display */
1194  1   GFMGR_Display (REST_GetFMgrContext());

1196  1   /* Flag that we are no longer sorting */
1197  1   REST_SetReReadInProgress (BOOL_FALSE);

1199  }
```



```
1201  /*****************************************************************
1202   * REST_GetSort
1203   *
1204   * Description:
1205   *   This routine will return the current sort type.
1206   *
1207   * Parameters:
1208   *   None.
1209   *
1210   * Returns:
1211   *   The type of sort to perform.
1212   *
1213   *****************************************************************/

1215  REST_SortType  REST_GetSort (void)
1216  1 {
1217  1   /* Return the current sort type */
1218  1   return (REST_CurrentSort);
1219  }
```

```
1221  /*****************************************************************
1222   * REST_UpdateViewOptions
1223   *
1224   * Description:
1225   *   This routine will update the display to the current display
1226   *   options (show hidden files, etc...).
1227   *
1228   * Parameters:
1229   *   None.
1230   *
1231   * Returns:
1232   *   None.
1233   *
1234   *****************************************************************/

1236  void      REST_UpdateViewOptions (void)
1237 1 {
1238 1   RestoreInfoPtr  info;      /* Pointer to walk the list with */
1239 1   RestoreInfoPtr  foundInfo;
                        /* Pointer to found object which is selected */

1241 1   /* No need to do anything if there is no current work item */
1242 1   if (currentWorkItemInfo != NULL)
1243 2   {
1244 2     /* Freeze the file manager display to avoid major flickering */
1245 2     GFMGR_FreezeDisplay (REST_GetFMgrContext());

1247 2     /* Flag that we are updating current children */
1248 2     REST_SetReReadInProgress (BOOL_TRUE);

1250 2     /* Walk the current list of objects */
1251 2     info = (RestoreInfoPtr) GFMGR_GetVeryFirstObject (
                                                  REST_GetFMgrContext());
1252 2     while (info != NULL)
1253 3     {
1254 3       /* Update the children */
1255 3       GFMGR_UpdateChildren (REST_GetFMgrContext(), info);

1257 3       /* Go to the next object */
1258 3       info = info->next;
1259 2     }

1261 2     /* Select the object which should be currently selected */
1262 2     foundInfo = REST_FindSelectionString ();
1263 2     if (foundInfo != NULL)
1264 3     {
1265 3       GFMGR_SelectObject (REST_GetFMgrContext(),
1266 3                           (GFMGR_Object)foundInfo,
1267 3                           BOOL_FALSE);
1268 3       REST_SetSelectionString (REST_GetFullName(foundInfo));
1269 2     }

1271 2     /* Update the file manager's display */
1272 2     GFMGR_Display (REST_GetFMgrContext());

1274 2     /* Flag that we are no longer updating current children */
1275 2     REST_SetReReadInProgress (BOOL_FALSE);

1277 2     /* Update the date buttons since we may have changed offsite
                                                              status */
1278 2     REST_UpdateDateButtons ();
1279 1   }
1280  }
```



```
1282  /*****************************************************************
1283   * REST_UpdateRemoveButtons
1284   *
1285   * Description:
1286   *   This routine will sensitize and desensize the remove buttons.
1287   *
1288   * Parameters:
1289   *   None.
1290   *
1291   * Returns:
1292   *   None.
1293   *
1294   *****************************************************************/

1296  void      REST_UpdateRemoveButtons (void)
1297 1 {

1299 1   /* If there is a restore in progress, don't update anything */
1300 1   if (REST_RestoreInProgress ())
1301 1     return;

1303 1   /* If there are any marked items, sensitize the remove all button */
1304 1   GUTIL_WGT_SetEnabled ((WgtPtr)REST_RestoreWin->ClearButton,
1305 1                         LBOX_GoAllocFirst (
                                    REST_RestoreWin->SelectedListBox));

1307 1   /* If there are any marked items sensitize the start button */
1308 1   GUTIL_WGT_SetEnabled ((WgtPtr)REST_RestoreWin->StartButton,
1309 1                         LBOX_GoAllocFirst (
                                    REST_RestoreWin->SelectedListBox));

1311 1   /* If there are any selected marked items,
                                        sensitize the remove button */
1312 1   GUTIL_WGT_SetEnabled ((WgtPtr)REST_RestoreWin->RemoveButton,
1313 1                         LBOX_GoSelAllocFirst (
                                    REST_RestoreWin->SelectedListBox));
1314  }
```

```
1316  /*****************************************************************
1317   * REST_MarkBackupItems
1318   *
1319   * Description:
1320   *   This routine will mark objects selected in the file manager list
                                                                      box.
1321   *
1322   * Parameters:
1323   *   None.
1324   *
1325   * Returns:
1326   *   None.
1327   *
1328   *****************************************************************/

1330  void      REST_MarkBackupItems (void)
1331  1 {
1332  1   RestoreInfoPtr info;                /* Next object to mark */
1333  1   BoolEnum       marked = BOOL_FALSE;
                                             /* Flag if anything was marked */
1334  1   long           numberBad = 0;       /* Number of bad files marked */
1335  1   long           numberMarked = 0;    /* Number of objects marked */

1337  1   /* This may take a while so display a wait cursor */
1338  1   WGT_UseWaitCursor ((WgtPtr)REST_RestoreWin);

1340  1   /* Select all items that are highlighted */
1341  1   info =(RestoreInfoPtr)GFMGR_GetFirstSelectedLBoxObject(
                                                  REST_GetFMgrContext());
1342  1   while (info != NULL)
1343  2   {
1344  2     /* If this is a different workitem */
1345  2     if (info->type == REST_WorkItem)
1346  3     {
1347  3       if (REST_VerifySelection (REST_GetFMgrContext(),
1348  3                                 info,
1349  3                                 BOOL_FALSE,
1350  3                                 BOOL_FALSE))
1351  4       {
1352  4         /*
1353  4          * Select the work item,
                                this causes it to be selected in the SArea
1354  4          * which isn't really the desired result,
                                                  but we have to have the
1355  4          * workitem in the SArea in order to continue
1356  4          */

1358  4         GFMGR_SelectObject (REST_GetFMgrContext(),
1359  4                             (GFMGR_Object)info,
1360  4                             BOOL_TRUE);
1361  3       }
1362  3       else
1363  4       {
1364  4         return;
1365  3       }
1366  2     }

1368  2     marked |= REST_MarkInfo (info, &numberMarked, &numberBad);

1370  2     /* get the next selected row */
1371  2     info=(RestoreInfoPtr)GFMGR_GetNextSelectedLBoxObject(
                                                  REST_GetFMgrContext());
1372  1   }

1374  1   /* If anything was marked,
```



```
                                        update the marked data based on the new list */
1375  1   if (marked)
1376  2   {
1377  2     REST_UpdateMarkedDataFromList (numberMarked, numberBad);
1378  1   }

1380  1   /* Update buttons based on what is selected in the LBox */
1381  1   SAREA_RedrawParts ((SAreaPtr)REST_RestoreWin->SelectedListBox);
1382  1   REST_LBoxSelectionNfy (NULL);
1383  1   REST_UpdateRemoveButtons ();

1385  1   /* Display the normal cursor again */
1386  1   WGT_UseCursor ((WgtPtr)REST_RestoreWin, CURS_Arrow());

1388  }
```

```
1390  /*****************************************************************
1391   * REST_UnmarkBackupItems
1392   *
1393   * Description:
1394   *   This routine will unmark objects selected in the file manager
                                                                  list box.
1395   *
1396   * Parameters:
1397   *   None.
1398   *
1399   * Returns:
1400   *   None.
1401   *
1402   *****************************************************************/

1404      void      REST_UnmarkBackupItems (void)
1405  1   {
1406  1     RestoreInfoPtr info;                    /* Next object to mark */
1407  1     long           numberBad = 0;
                                              /* Number of bad files marked */
1408  1     long           numberMarked = 0;
                                             /* Number of objects marked */
1409  1     BoolEnum       unmarked = BOOL_FALSE;
                                          /* Flag if anything was unmarked */

1411  1     /* This may take a while so display a wait cursor */
1412  1     WGT_UseWaitCursor ((WgtPtr)REST_RestoreWin);

1414  1     /* Un-Select all items that are highlighted */
1415  1     info =(RestoreInfoPtr)GFMGR_GetFirstSelectedLBoxObject(
                                                  REST_GetFMgrContext());
1416  1     while (info != NULL)
1417  2     {
1418  2       /* Update the overall unmarked flag */
1419  2       unmarked |= REST_UnmarkInfo (info, &numberMarked, &numberBad);

1421  2       /* get the next selected row */
1422  2       info=(RestoreInfoPtr)GFMGR_GetNextSelectedLBoxObject(
                                                  REST_GetFMgrContext());
1423  1     }

1425  1     /* If anything was unmarked,
                             update the marked data based on the new list */
1426  1     if (unmarked)
1427  2     {
1428  2       REST_UpdateMarkedDataFromList (numberMarked, numberBad);
1429  1     }

1431  1     /* Update buttons based on what is selected in the LBox */
1432  1     SAREA_RedrawParts ((SAreaPtr)REST_RestoreWin->SelectedListBox);
1433  1     REST_LBoxSelectionNfy (NULL);
1434  1     REST_UpdateRemoveButtons ();

1436  1     /* Display the normal cursor again */
1437  1     WGT_UseCursor ((WgtPtr)REST_RestoreWin, CURS_Arrow());
1438    }
```



```
1440  /*****************************************************************
1441   * REST_SetSearchVisiblity
1442   *
1443   * Description:
1444   *   This routine will set the search button visibility to the given
1445   *   value.
1446   *
1447   * Parameters:
1448   *   visible (I) - Flag if the button should be visible.
1449   *
1450   * Returns:
1451   *   None.
1452   *
1453   *****************************************************************/

1455      void REST_SetSearchVisiblity (BoolEnum visible)
1456  1   {
1457  1     BoolEnum isVisible;

1459  1     /* If there is a restore in progress, don't update anything */
1460  1     if (REST_RestoreInProgress ())
1461  1       return;

1463  1     /* Based on the incoming value and the searchableness of the current
                                                                        WI */
1464  1     if (visible &&
1465  1         (currentWorkItemInfo != NULL) &&
1466  1         (currentWorkItemInfo->restoreObject != NULL))
1467  2     {
1468  2       /* Make sure the current work item is searchable */
1469  2       if (EDMRST_IsObjectSearchable (GREST_Handle,
1470  2                                      currentWorkItemInfo->restoreObject,
1471  2                                      &isVisible) != E_SUCCESS)
1472  2         isVisible = BOOL_FALSE;    /* on error, set not visible */
1473  1     }
1474  1     else
1475  2     {
1476  2       /* Not visible */
1477  2       isVisible = BOOL_FALSE;
1478  1     }

1480  1     /* Set the visibility of the search button */
1481  1     GUTIL_WGT_SetEnabled ((
                          WgtPtr)REST_RestoreWin->SearchButton, isVisible);
1482    }
```

```
1484  /*****************************************************************
1485   * REST_UpdatePartialButton
1486   *
1487   * Description:
1488   *   This routine will update the allow partials button visibility
                                                                    based
1489   *   on what is allowed for the current work item.
1490   *
1491   * Parameters:
1492   *   None
1493   *
1494   * Returns:
1495   *   None.
1496   *
1497   *****************************************************************/

1499  void REST_UpdatePartialButton ( void )
1500  1 {
1501  1   BoolEnum isSymmOK;

1503  1   /* If there is a restore in progress, don't update anything */
1504  1   if (REST_RestoreInProgress ())
1505  1     return;

1507  1   /* Based on the current WI determin if SC restore is OK */
1508  1   if ((currentWorkItemInfo != NULL) &&
1509  1       (currentWorkItemInfo->restoreObject != NULL))
1510  2   {
1511  2     /* Make sure the current work item is searchable */
1512  2     if (EDMRST_GetSymmRestoreOption (GREST_Handle,
1513  2
                                       currentWorkItemInfo->restoreObject,
1514  2                                    &isSymmOK) != E_SUCCESS)
1515  3     {
1516  3       /* on error, allow SC restore */
1517  3       isSymmOK = BOOL_TRUE;
1518  2     }
1519  1   }
1520  1   else
1521  2   {
1522  2     /* Allow SC to be visible,
                                  it will be updated more accurately later */
1523  2     isSymmOK = BOOL_TRUE;
1524  1   }

1526  1   if (!isSymmOK)
1527  2   {
1528  2     /* Not OK to use symm connect,
                                      unselect the allow partion button */
1529  2     TBUT_Unselect ((TButPtr)REST_RestoreWin->AllowPartialButton);
1530  1   }

1532  1   /* Set the visibility of the partion button  */
1533  1   GUTIL_WGT_SetEnabled ((
                    WgtPtr)REST_RestoreWin->AllowPartialButton, isSymmOK);
1534  }
```



```
1536  /*****************************************************************
1537   * REST_DisplaySearch
1538   *
1539   * Description:
1540   *   This routine will display the restore search window.
1541   *
1542   * Parameters:
1543   *   None.
1544   *
1545   * Returns:
1546   *   None.
1547   *
1548   *****************************************************************/

1550  void      REST_DisplaySearch (void)
1551  1 {
1552  1   RestoreInfoPtr info;
                                  /* Object selected in the file manager */
1553  1   Char           startString[MAX_STRING_LENGTH];
                                        /* Starting directory string */

1555  1   /* Get the current directory from the file manager */
1556  1   info = (RestoreInfoPtr)GFMGR_GetFirstSelectedObject (
                                                    REST_GetFMgrContext());

1558  1   /* If there is a current directory, get its name */
1559  1   if (info->restoreObject != NULL)
1560  2   {
1561  2     /* If it really is a directory get the full name */
1562  2     if (info->type == REST_Directory)
1563  3     {
1564  3       STR_Cpy (startString,
1565  3                EDMRST_GetObjectFullName (
                                      GREST_Handle, info->restoreObject));
1566  2     }

1568  2     /* If this is a client or a work-item,
                                           get the name of its first child */
1569  2     else if ((info->children != NULL) &&
1570  2              (info->children->next == NULL) &&
1571  2              (info->children->type == REST_Directory))
1572  3     {
1573  3       STR_Cpy (startString,
1574  3                EDMRST_GetObjectFullName (GREST_Handle,
1575  3
                                           info->children->restoreObject));
1576  2     }

1578  2     /* Else just start at the root directory */
1579  2     else
1580  3     {
1581  3       STR_Cpy (startString, "/");
1582  2     }
1583  1   }

1585  1   /* Else there is nowhere to start */
1586  1   else
1587  2   {
1588  2     STR_Cpy (startString, "");
1589  1   }

1591  1   /* Put up the search window */
1592  1   REST_SearchDisplay (startString);
1593  }
```

```
1595  /*****************************************************************
1596   * REST_TimedOut
1597   *
1598   * Description:
1599   *   This routine is called when the user has been idle too long and
1600   *   we want to time out the process.  Rather than exit the process
1601   *   all together,
                    we want to reset the user back to the initial state.
1602   *
1603   * Parameters:
1604   *   code (I) - The notification received.
1605   *
1606   * Returns:
1607   *   None.
1608   *
1609   *****************************************************************/

1611  void REST_TimedOut (ClientPtr clientData)
1612 1 {
1613 1   int               countDown;    /* The current countdown value */
1614 1   GALERT_WinHandle  synchHandle;  /* Handle to the alert dialog */
1615 1   Char              outputString[MAX_STRING_LENGTH];
                                                  /* String to display */

1617 1   /* Re-add the timeout incase the user still doesn't do anything */
1618 1   EVENT_StartCBackAlarm (REST_TimedOut, (
                                  ClientPtr)clientData, TIMEOUT_DELAY);

1620 1   /* We only care if we have a work-item open,
                                        and a restore is not running */
1621 1   if ((currentWorkItemInfo != NULL) &&
1622 1       (!REST_RestoreInProgress()) &&
1623 1       (!REST_SearchInProgress()))
1624 2   {

1626 2     /* Show the restore window if it is iconified */
1627 2     if (!WIN_IsOpen ((WinPtr)REST_RestoreWin))
1628 2       WIN_Show ((WinPtr)REST_RestoreWin);

1630 2     /* Format a warning to countdown before re-initing */
1631 2     STR_Sprintf (outputString,
1632 2                  REST_GetErrorString (REST_TIMEOUT_FORMAT_WARNING),
1633 2                  TIMEOUT_WARNING_SECONDS);

1635 2     /* Pop up the warning dialog */
1636 2     synchHandle = GALERT_DisplaySynchronousWait
1637 2                                   ((WinPtr)REST_RestoreWin,
1638 2                                    REST_GetErrorString (
                                                  REST_WARNING_INDEX),
1639 2                                    GICON_GetWarning(),
1640 2                                    outputString,
1641 2                                    BOOL_TRUE);

1643 2     /* Countdown the seconds before re-initing */
1644 2     for (countDown = TIMEOUT_WARNING_SECONDS;
1645 2          (!GALERT_IsCancelled (synchHandle) && (countDown > 0));
1646 2          countDown--)
1647 3     {
1648 3       /* The user hasn't cancelled,
                                 display the number of seconds remaining */
1649 3       STR_Sprintf (outputString,
1650 3                    REST_GetErrorString (REST_TIMEOUT_FORMAT_WARNING),
1651 3                    countDown);
1652 3       GALERT_UpdateMessage (synchHandle, outputString);
```



```
1654 3       /* Wait one second */
1655 3       sleep (1);
1656 2     }

1658 2     /* Check to see if the user cancelled */
1659 2     if (!GALERT_IsCancelled (synchHandle))
1660 3     {
1661 3       /* The user didn't cancel, clear this session and redisplay */
1662 3       REST_ClearSession (BOOL_TRUE);
1663 3       REST_Display ();

1665 3       /* Remove the wait dialog */
1666 3       GALERT_CancelSynchDialog (synchHandle);

1668 3       /* Display an alert dialog telling the user we timed out */
1669 3       GALERT_DisplayError ((WinPtr)REST_RestoreWin,
1670 3                            REST_GetErrorString (REST_WARNING_INDEX),
1671 3                            GICON_GetWarning(),
1672 3                            REST_GetErrorString (REST_TIMEOUT_ERROR));

1674 2     }
1675 2     else
1676 3     {
1677 3       /* Remove the wait dialog */
1678 3       GALERT_CancelSynchDialog (synchHandle);
1679 2     }
1680 1   }
1681 1   else
1682 2   {
1683 2     /*
1684 2      * Keep the restore engine alive by telling it we're still here
1685 2      */

1687 2     EDMRST_Ping (GREST_Handle);
1688 1   }

1690  }
```

```
1692  /*****************************************************************
1693   * REST_NotifyReceived
1694   *
1695   * Description:
1696   *   This routine keeps track of the time between user actions.  If
1697   *   the user is idle for too long a time, the Restore API will be
1698   *   reset so that locks on Work-Items are not held which would cause
1699   *   eb processes to hang.
1700   *
1701   * Parameters:
1702   *   code (I) - The notification received.
1703   *
1704   * Returns:
1705   *   None.
1706   *
1707   *****************************************************************/

1709  void      REST_NotifyReceived (WinNfyEnum code)
1710 1{

1712 1   if ((code == WIN_NFYMOUSEMOVE) ||
1713 1       (code == WIN_NFYMOUSECLICK) ||
1714 1       (code == WIN_NFYMOUSEBUTTONRELEASE) ||
1715 1       (code == WIN_NFYKEYACCEL) ||
1716 1       (code == WIN_NFYKEYCIRC) ||
1717 1       (code == WIN_NFYKEYCHAR) ||
1718 1       (code == WIN_NFYGAINFOCUS))
1719 2   {
1720 2     if (timerData != NULL)
1721 2       EVENT_StopCBackAlarm (timerData);
1722 2     else
1723 2       timerData = GUTIL_Malloc (sizeof(Int));

1725 2     EVENT_StartCBackAlarm (REST_TimedOut, (
                                    ClientPtr)timerData, TIMEOUT_DELAY);
1726 1   }
1727  }
```



```
1729  /*****************************************************************
1730   * REST_DisplayContextHelp
1731   *
1732   * Description:
1733   *   This routine will display context sensitive help for the give
1734   *   widget.
1735   *
1736   * Parameters:
1737   *   contextWiget (I) - The current widget to display help for.
1738   *
1739   * Returns:
1740   *   None.
1741   *
1742   *****************************************************************/

1744  void REST_DisplayContextHelp (WgtPtr contextWiget)
1745 1{
1746 1   WgtPtr         focusWgt;
1747 1   int            helpID = REST_MAIN;
1748 1   BoolEnum       found = BOOL_FALSE;

1750 1   focusWgt = contextWiget;
1751 1   while ((!found) && (focusWgt != NULL))
1752 2   {
1753 2     if (focusWgt == (WgtPtr)REST_RestoreWin->DataAvailablePanel)
1754 3     {
1755 3       helpID = REST_DATA_AVAILABLE;
1756 3       found = BOOL_TRUE;
1757 2     }
1758 2     else if (focusWgt == (WgtPtr)REST_RestoreWin->TabsPanel)
1759 3     {
1760 3       if (WGT_IsVisible((WgtPtr)REST_RestoreWin->ViewOptionsPanel))
1761 3         helpID = REST_VIEW_OPTIONS;
1762 3       else if (WGT_IsVisible((
                                    WgtPtr)REST_RestoreWin->MarkSummaryPanel))
1763 3         helpID = REST_MARK_SUMMARY;
1764 3       else if (WGT_IsVisible((WgtPtr)REST_RestoreWin->MediaPanel))
1765 3         helpID = REST_MEDIA;
1766 3       else
1767 3         helpID = REST_MAIN;
1768 3       found = BOOL_TRUE;
1769 2     }
1770 2     else
1771 3     {
1772 3       focusWgt = (WgtPtr) WGT_GetPanel (focusWgt);
1773 2     }
1774 1   }
1775 1   EDMHELP_Display ((WinPtr)REST_RestoreWin, REST_MODID, helpID);
1776  }
```

```
/*****************************************************************
 * restCalMgr.c
 *
 *
 * Copyright 1996 by Epoch Systems, Inc.
 *
 *
 * Mission Statement:
 *   This file contains the functions necessary for the EDM Calendar
                                                                window.
 *
 * Required includes:
 *   None
 *
 * Compile-Time Options:
 *   N/A
 *
 *
 * RCS Information:
 *     $RCSfile$
 *     $Revision$
 *     $Date$
 *****************************************************************/

#define  ERR_LIB          RESTORE

#include <esl/c_portable.h>

#include <respub.h>
#include <cboxpub.h>
#include <panelpub.h>
#include <tbutpub.h>
#include <winpub.h>
#include <drawpub.h>

#include "eerrno/e_errno.h"
#include <restore/restore_api.h>

#include "restore.h"
#include "restoreP.h"
#include "restUtils.h"
#include "gutil/winUtils.h"
#include "restCalendar.h"
#include "gutil/timeUtils.h"
#include "gutil/cboxUtils.h"
#include "gutil/guiUtils.h"
#include "gutil/iconUtils.h"
#include "gutil/alertMgr.h"

/*************
 * Constants *
 *************/

#define SMALL_STRING_LENGTH  16
#define MEDIUM_STRING_LENGTH 32

#define DAY_LABEL_OFFSET     3
#define DAY_LABEL_HEIGHT    20
#define MAX_TIME_SLOTS      24

#define TIME_BUFFER_LENGTH  64

#define DATE_CBOX_WIDTH    140
#define DATE_CBOX_HEIGHT    90

#define DATE_TEXT_HEIGHT    21
```



```
#define MAX_DAYS_PER_MONTH  31

#define BORDER_OFFSET        5
#define MARGIN_WIDTH        15

/********************
 * Global Variables *
 ********************/

/* Store the currently displayed month and year */
static int displayedMonth;
static int displayedYear;

/* Store the current time for the displayed month */
static struct tm displayedMonthTime;

/* Store the currently selected time */
static time_t currentSelectedTime;

/* Store the currently selected day, month, and year */
static int selectedDay = -1;
static int selectedMonth;
static int selectedYear;

/* Arrays kept for the backup times for the current month */
static time_t  backupTimes[MAX_DAYS_PER_MONTH][MAX_TIME_SLOTS];
static int     backupTimesCount[MAX_DAYS_PER_MONTH];

/*
 * Store the first and last backup times,
                                        so we know when there are previous
 * and next backups.
 */

/* Array of Combo Boxes (
                          one per day) to show list of times for that day */
static CBoxPtr backupTimeBox[MAX_DAYS_PER_MONTH];

/* Array of Text Edits (one per day) to show the time for that day */
static TEdPtr backupTimeText[MAX_DAYS_PER_MONTH];

/* Flags to determine Initialization status */
static BoolEnum boxesInited = BOOL_FALSE;
static BoolEnum restoreCalendarInited = BOOL_FALSE;

static BoolEnum REST_CalTerminated = BOOL_FALSE;

/*
 * Hack flag to avoid setting the selected time when a CurSelect is
                                                                done to
 * display the first time for a day (
                                       which is not the real time selected)
 */

static BoolEnum selectionByCode = BOOL_FALSE;

/* Forward Declaration */
static void REST_UpdateDisplayedDate (void);

/*****************************************************************
 * REST_CalCreateDateBox
 *
 * Description:
 *   This routine will create a combo box to be used for displaying
 *   multiple restoral times for a single day.
```

```
 *
 * Parameters:
 *   None.
 *
 * Returns:
 *   The new Combo Box
 *
 ************************************************************************/

static CBoxPtr REST_CalCreateDateBox (void)
{
  CBoxPtr     newBox;  /* The new combo box created */
  ColorPtr    fgColor; /* The color to set the foreground to */
  ColorPtr    bgColor; /* The color to set the background to */
  PenPtr      pen;     /* The pen to use for the combo box */
  FontPtr     font;    /* The font to use for the combo box */

  /*
   * Create the combo box and set the defaults
   */

  newBox = CBOX_Create ();
  CBOX_SetLabel (newBox, "");
  CBOX_SetSubClass (newBox, CBOX_TYPEDROPDOWNLIST);

  /* Set the resizing so that it doesn't resize at all */
  WGT_SetRzFlags ((WgtPtr)newBox,
                  WGT_RESIZEKEEPWIDTH | WGT_RESIZEKEEPHEIGHT |
                  WGT_RESIZEKEEPLEFT | WGT_RESIZEKEEPRIGHT);

  /*
   * Attempt to get each 'eden' default resource and set it
   */

  if ((fgColor = (ColorPtr)RES_Find ("eden", "CBoxFgColor")) != NULL)
    WGT_SetFgColor ((WgtPtr)newBox, fgColor);

  if ((bgColor = (ColorPtr)RES_Find ("eden", "CBoxBgColor")) != NULL)
    WGT_SetBgColor ((WgtPtr)newBox, bgColor);

  if ((pen = (PenPtr)RES_Find ("eden", "CBoxPen")) != NULL)
    WGT_SetPen ((WgtPtr)newBox, pen);

  if ((font = (FontPtr)RES_Find ("eden", "CBoxFont")) != NULL)
    WGT_SetFont ((WgtPtr)newBox, font);

  /* Return the new box */
  return (newBox);
}
```



```

/************************************************************************
 * REST_CalCreateDateText
 *
 * Description:
 *   This routine will create a TEd to be used for displaying
 *   a singl restoral time for a day.
 *
 * Parameters:
 *   None.
 *
 * Returns:
 *   The new Text Area
 *
 ************************************************************************/

static TEdPtr REST_CalCreateDateText (void)
{
  TEdPtr      newText;  /* The new text area created */
  ColorPtr    fgColor;  /* The color to set the foreground to */
  ColorPtr    bgColor;  /* The color to set the background to */
  PenPtr      pen;      /* The pen to use for the combo box */
  FontPtr     font;     /* The font to use for the text area */

  /*
   * Create the text area and set the defaults
   */

  newText = TED_Create ();
  TED_SetJustif (newText, DRAW_JUSTCENTER);
  TED_SetOptFlags (newText, TED_OPTOUTPUTONLY);

  /* Set the resizing so that it doesn't resize at all */
  WGT_SetRzFlags ((WgtPtr)newText,
                  WGT_RESIZEKEEPWIDTH | WGT_RESIZEKEEPHEIGHT |
                  WGT_RESIZEKEEPLEFT | WGT_RESIZEKEEPRIGHT);

  /*
   * Attempt to get each 'eden' default resource and set it
   */

  if ((fgColor = (ColorPtr)RES_Find (
                                     "eden", "OutputFgColor")) != NULL)
    WGT_SetFgColor ((WgtPtr)newText, fgColor);

  if ((bgColor = (ColorPtr)RES_Find (
                                     "eden", "OutputBgColor")) != NULL)
    WGT_SetBgColor ((WgtPtr)newText, bgColor);

  if ((pen = (PenPtr)RES_Find ("eden", "OutputPen")) != NULL)
    WGT_SetPen ((WgtPtr)newText, pen);

  if ((font = (FontPtr)RES_Find ("eden", "OutputFont")) != NULL)
    WGT_SetFont ((WgtPtr)newText, font);

  /* Return the new text area */
  return (newText);
}
```

```
234  /*****************************************************************
235   * REST_CalSetBoxPos
236   *
237   * Description:
238   *   This routine will set the position for the combo box and TEd
239   *   for the given day given the coordinates for the day's entire box.
240   *
241   * Parameters:
242   *   dayNumber (I) - The day to update the combo box and TEd for.
243   *   dayOri    (I) - The origin coordinates for the day's box.
244   *   dayExt    (I) - The extent coordinates for the day's box.
245   *
246   * Returns:
247   *   None.
248   *
249   *****************************************************************/

251  static void REST_CalSetBoxPos (Int        dayNumber,
252                                 Point16Rec dayOri,
253                                 Point16Rec dayExt)
254 1 {
255 1   Rect16Rec    box;  /* Bounding box for the combo box and TEd */

257 1   /* Set the width to leave an offset on each side */
258 1   box.Ext.x = dayExt.x - (2 * DAY_LABEL_OFFSET);

260 1   /* If this is too wide use the max width */
261 1   if (box.Ext.x > DATE_CBOX_WIDTH)
262 2   {
263 2     box.Ext.x = DATE_CBOX_WIDTH;
264 1   }

266 1   /* Center the box, horizontally */
267 1   box.Ori.x = dayOri.x + ((dayExt.x - box.Ext.x) / 2);

269 1   /* Set the height of the box */
270 1   box.Ext.y = DATE_CBOX_HEIGHT;

272 1   /* Center the box, vertically */
273 1   box.Ori.y = dayOri.y + ((dayExt.y - DATE_TEXT_HEIGHT ) / 2);

275 1   /* Set the combo box's bounding box */
276 1   WGT_SetBox ((WgtPtr)backupTimeBox[dayNumber], &box);

278 1   /* Set the text area's bounding box */
279 1   box.Ext.y = DATE_TEXT_HEIGHT;
280 1   WGT_SetBox ((WgtPtr)backupTimeText[dayNumber], &box);

282  }
```



```
284  /*****************************************************************
285   * REST_GetDayWidthHeight
286   *
287   * Description:
288   *   This routine will determine the current width and height of a day
289   *   in the calendar window.
290   *
291   * Parameters:
292   *   width (O)  - The width of the days.
293   *   height (O) - The height of the days.
294   *
295   * Returns:
296   *   None.
297   *
298   *****************************************************************/

300  void REST_GetDayWidthHeight (Int  *width,
301                               Int  *height)
302 1 {
303 1   int          numberDays;
                                /* Number of days in the displayed month */
304 1   int          numberWeeks;
                               /* Number of weeks in the displayed month */
305 1   Point16Ptr   panelExt;    /* Extents of the panel */
306 1   Int          panelWidth;  /* The drawable panel width */
307 1   Int          panelHeight; /* The drawable panel height */

309 1   /* Determine the number of days and weeks in the displayed month */
310 1   numberDays = GTIME_DaysPerMonth (displayedMonth, displayedYear);
311 1   numberWeeks = GTIME_NumberWeeks (
                                  displayedMonthTime.tm_wday, numberDays);

313 1   /* Get the current extents of the drawing panel */
314 1   panelExt = (Point16Ptr) WGT_GetExt ((
                                WgtPtr)REST_CalendarWindow->CalendarPanel);
315 1   panelWidth = panelExt->x - (2 * MARGIN_WIDTH);
316 1   panelHeight = panelExt->y - 1;

318 1   /* Get the width and height of each day */
319 1   *width = panelWidth / DAYS_PER_WEEK;
320 1   *height = (panelHeight - DAY_LABEL_HEIGHT) / numberWeeks;

322  }
```

```
 324  /*********************************************************************
 325   * REST_DrawSelectedDay
 326   *
 327   * Description:
 328   *   This routine will draw the border and text number for the
                                                                  currently
 329   *   selected day.
 330   *
 331   * Parameters:
 332   *   drawSelected - Flag if the day should still be drawn selected.
 333   *
 334   * Returns:
 335   *   None.
 336   *
 337   *********************************************************************/

 339  static void REST_DrawSelectedDay (BoolEnum drawSelected)
 340 1 {
 341 1   Rect16Rec    rect;         /* Rectangle to draw */
 342 1   int          dayNumber;    /* Day number (0-6) of the selected day */
 343 1   int          weekNumber;   /* Week number of the selected day */
 344 1   Int          dayWidth;     /* The width of a day box */
 345 1   Int          dayHeight;    /* The height of a day box */

 347 1   /* Only draw if there is a selected day */
 348 1   if (selectedDay >= 0)
 349 2   {

 351 2     /* Only draw if the selected day is in the current month/year */
 352 2     if ((selectedMonth == displayedMonth) && (
                                        selectedYear == displayedYear))
 353 3     {

 355 3       /* Get the width and height of the day */
 356 3       REST_GetDayWidthHeight (&dayWidth, &dayHeight);

 358 3       /* Determine the day number (0-6) */
 359 3       dayNumber = ((selectedDay % DAYS_PER_WEEK) +
 360 3                    displayedMonthTime.tm_wday) % DAYS_PER_WEEK;

 362 3       /* Determine the week number */
 363 3       weekNumber = (
                 selectedDay + displayedMonthTime.tm_wday) / DAYS_PER_WEEK;

 365 3       /* Calculate the bounding box for the selected day */
 366 3       rect.Ori.x = MARGIN_WIDTH + dayWidth * dayNumber;
 367 3       rect.Ori.y = DAY_LABEL_HEIGHT + (dayHeight * weekNumber);
 368 3       rect.Ext.x = dayWidth;
 369 3       rect.Ext.y = dayHeight;

 371 3       /* IF the day is still selected, draw the border */
 372 3       if (drawSelected)
 373 4       {
 374 4         /* Draw in one pixel all the way around,
                                                 to preserve the border */
 375 4         rect.Ori.x++;
 376 4         rect.Ori.y++;
 377 4         rect.Ext.x--;
 378 4         rect.Ext.y--;

 380 4         /* Draw a thick border around the selected day */
 381 4         DRAW_SetPen((
                   WgtPtr)REST_CalendarWindow->CalendarPanel, PEN_Solid2());
 382 4         DRAW_SetColors ((WgtPtr)REST_CalendarWindow->CalendarPanel,
 383 4                         COLOR_Black(),
```



```
 384 4                         COLOR_Transparent());
 385 4         DRAW_Rect ((WgtPtr)REST_CalendarWindow->CalendarPanel, &rect);

 387 3       }

 389 3       /* IF not still selected, just invalidate the region */
 390 3       else
 391 4       {
 392 4         /* Redraw a large enough rectangle to cover the entire day */
 393 4         rect.Ori.x -= BORDER_OFFSET;
 394 4         rect.Ori.y -= BORDER_OFFSET;
 395 4         rect.Ext.x += (2 * BORDER_OFFSET);
 396 4         rect.Ext.y += (2 * BORDER_OFFSET);
 397 4         DRAW_InvalRect ((
                     WgtPtr)REST_CalendarWindow->CalendarPanel, &rect);
 398 3       }
 399 2     }
 400 1   }
 401  }
```

```
403  /*****************************************************************
404   * REST_CalendarSelectDay
405   *
406   * Description:
407   *   This routine will select the day at the location of the current
408   *   mouse click.
409   *
410   * Parameters:
411   *   None.
412   *
413   * Returns:
414   *   BOOL_TRUE  - If the day was successfully selected
415   *   BOOL_FALSE - otherwise
416   *
417   *****************************************************************/

419  BoolEnum REST_CalendarSelectDay (void)
420 1 {
421 1   Point16Rec loc;                    /* Location of the mouse click */
422 1   Rect16Ptr  wgtBox;
                                         /* Bounding box for the combo box */
423 1   Rect16Rec  box;
                                      /* Box for the combo box with border */
424 1   Point16Rec penSize;                /* Size of the combo box's pen */
425 1   BoolEnum   inTimeBox = BOOL_FALSE;
                                      /* Flag if click was in the combo box */
426 1   int        dayNumber;
                                      /* Number of the day column (0-6) */
427 1   int        weekNumber;            /* Number of the week (row) */
428 1   int        numberDays;
                                   /* Number of days in the displayed month */
429 1   int        boxNumber;             /* Total box number */
430 1   int        date;                  /* Day of the month */
431 1   BoolEnum   returnStatus;
                                   /* Flag if the click was in a valid date */
432 1   Int        index;
                                      /* Index for time on selected day */
433 1   Int        dayWidth;              /* The width of a day box */
434 1   Int        dayHeight;             /* The height of a day box */

436 1   /* Get the location of the click */
437 1   EVENT_QueryWgtLoc((WgtPtr)REST_CalendarWindow->CalendarPanel, &loc);

439 1   /* Get the width and height of the day */
440 1   REST_GetDayWidthHeight (&dayWidth, &dayHeight);

442 1   /* Get the day (column) and week (row) */
443 1   dayNumber  = ((loc.x - MARGIN_WIDTH)/ dayWidth) + 1;
444 1   weekNumber = (loc.y - DAY_LABEL_HEIGHT) / dayHeight;

446 1   /* Validate the location against the drawing area */
447 1   if ((loc.y < DAY_LABEL_HEIGHT) ||
448 1       (loc.x < MARGIN_WIDTH) ||
449 1       (loc.x > (MARGIN_WIDTH + (dayWidth * DAYS_PER_WEEK))) ||
450 1       (dayNumber > DAYS_PER_WEEK))
451 2   {
452 2     returnStatus = BOOL_FALSE;
453 1   }
454 1   else
455 2   {
456 2     /* Get the overall box number */
457 2     boxNumber = dayNumber + (weekNumber * DAYS_PER_WEEK);

459 2     /* Determine the date */
460 2     date = boxNumber - displayedMonthTime.tm_wday;
```



```
462 2     /* Validate the date,
                      make sure at least one backup was done on that day */
463 2     numberDays = GTIME_DaysPerMonth (displayedMonth, displayedYear);
464 2     if ((date > 0) && (date <= numberDays) && (
                                              backupTimesCount[date-1] > 0))
465 3     {

467 3       /*
468 3        * Don't include clicks that are inside the combo box
469 3        */

471 3       /* If there is more than one backup on the date */
472 3       if (backupTimesCount[date-1] > 1)
473 4       {
474 4         /* Get the sizes */
475 4         PEN_QuerySize (WGT_GetPen (backupTimeBox[date - 1]), &penSize);
476 4         wgtBox = (Rect16Ptr) WGT_GetBox (backupTimeBox[date - 1]);

478 4         /* Get the overall area of the combo box */
479 4         box.Ori.x = wgtBox->Ori.x - penSize.x;
480 4         box.Ori.y = wgtBox->Ori.y - penSize.y;
481 4         box.Ext.x = wgtBox->Ext.x + (4 * penSize.x);
482 4         box.Ext.y = wgtBox->Ext.y + (4 * penSize.y);

484 4         /* Determine if the click was in the box */
485 4         inTimeBox = RECT_ContainsPoint(&box, &loc);
486 3       }

488 3       /* Continue if the click was not in a combo box */
489 3       if (!inTimeBox)
490 4       {

492 4         /* Redraw the old selected day (has affect of unselecting) */
493 4         REST_DrawSelectedDay (BOOL_FALSE);

495 4         /* Select the date that was clicked on */
496 4         selectedDay = date - 1;
497 4         selectedMonth = displayedMonth;
498 4         selectedYear = displayedYear;

500 4         /* Set the currently selected time */
501 4         if (backupTimesCount[date-1] == 1)
502 5         {
503 5           /* Obvious choice, there's only one */
504 5           currentSelectedTime = backupTimes[selectedDay][0];
505 4         }
506 4         else
507 5         {
508 5           /* Get the currently chosen index (
                                                 that's what the user sees) */
509 5           index = (Int) CBOX_ChosenGetId (backupTimeBox[selectedDay]);
510 5           currentSelectedTime = backupTimes[selectedDay][index];
511 4         }

513 4         /* Draw the newly selected day */
514 4         REST_DrawSelectedDay (BOOL_TRUE);

516 4         /* Flag that the selection was accepted */
517 4         returnStatus = BOOL_TRUE;
518 3       }
519 3       else
520 4       {
521 4         /* Flag that the selection was rejected */
522 4         returnStatus = BOOL_FALSE;
523 3       }
```

```
524 2      }
525 2      else
526 3      {
527 3        /* Flag that the selection was rejected */
528 3        returnStatus = BOOL_FALSE;
529 2      }
530 1    }

532 1    /* Return the status */
533 1    return (returnStatus);
534    }
```



```
536  /*****************************************************************
537   * REST_BackupExistsInMonth
538   *
539   * Description:
540   *   This routine will determine if a backup exists in the given month
541   *   for the given year.
542   *
543   * Parameters:
544   *   month (I) - Index of month to check
545   *   year  (I) - Year
546   *
547   * Returns:
548   *   BOOL_TRUE  - If a backup was done in the given month/year
549   *   BOOL_FALSE - otherwise
550   *
551   *****************************************************************/

553  static BoolEnum REST_BackupExistsInMonth (Int     month,
554                                            Int     year)
555 1 {
556 1    int         startDay = 1;              /* First day of the month */
557 1    time_t      endMonthTime;
                                              /* Last time value for the month */
558 1    time_t      startMonthTime;
                                             /* First time value for the month */
559 1    struct tm   thisTime;                  /* Time for current query */
560 1    long        cookie = INIT_COOKIE;      /* Ahh, the magic cookie */
561 1    short       numberEntries;
                                              /* Number of entries returned */
562 1    time_t     *times;
                                             /* Array of valid backup times */
563 1    eerrno_ty   eerrno;                    /* Error code returned */
564 1    BoolEnum    backupsExist = BOOL_FALSE;/* Flag if backups exist */
565 1    u_long      flags;

567 1    if (TBUT_GetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton))
568 1      flags = BACKUP_SELECTION_FLAG_PARTIAL_OK;
569 1    else
570 1      flags = BACKUP_SELECTION_FLAG_COMPLETE_ONLY;

572 1    /* Update the month and year in case of bad range */
573 1    GTIME_UpdateDate (&startDay, &month, &year, BOOL_FALSE);

575 1    /* Get the start time of the month */
576 1    GTIME_GetTimeTm (startDay,
577 1                     month,
578 1                     year,
579 1                     0,
580 1                     0,
581 1                     &thisTime);
582 1    startMonthTime = mktime (&thisTime);

584 1    /* Get the End time of the month */
585 1    GTIME_GetTimeTm (GTIME_DaysPerMonth(month, year),
586 1                     month,
587 1                     year,
588 1                     HOURS_PER_DAY - 1,
589 1                     MINUTES_PER_HOUR - 1,
590 1                     &thisTime);
591 1    endMonthTime = mktime (&thisTime);

593 1    /* Atttempt to get 1 time during the month */
594 1    times = (time_t *) GUTIL_Malloc (sizeof(time_t));
595 1    if ((eerrno = EDMRST_GetAllBackupTimes (GREST_Handle,
596 1                                            startMonthTime,
```

```
597  1                                          endMonthTime,
598  1                                          flags,
599  1                                          1,
600  1                                          times,
601  1                                          &numberEntries,
602  1                                          &cookie)) == E_SUCCESS)
603  2    {

605  2      /* If we got any back, then there are backups in that month */
606  2      if (numberEntries > 0)
607  2        backupsExist = BOOL_TRUE;
608  1    }
609  1    GUTIL_Free (times);

611  1    return (backupsExist);
612       }
```



```
614   /*****************************************************************
615    * REST_SetPreviousBackupInMonth
616    *
617    * Description:
618    *   This routine will select the first backup (chronologically) done
619    *   before the given day of the current month and year.
620    *
621    * Parameters:
622    *   day (IO) - the day to start at, updated to found day (if any)
623    *
624    * Returns:
625    *   BOOL_TRUE  - If a previous backup was found and selected
626    *   BOOL_FALSE - otherwise
627    *
628    *****************************************************************/

630   static BoolEnum REST_SetPreviousBackupInMonth (Int *day)
631  1{
632  1   BoolEnum found = BOOL_FALSE;
                                    /* Flag if we found a previous backup */
633  1   Int      tempDay;                  /* Temporary day holder */
634  1   Int      index;                    /* Loop index */
635  1   BoolEnum returnStatus = BOOL_FALSE;/* Status to return */

637  1   /* Find the previous day with a backup */
638  1   tempDay = *day - 1;
639  1   while ((!found) && (tempDay >= 0))
640  2   {
641  2     if (backupTimesCount[tempDay] > 0)
642  3     {
643  3       /* This is the day */
644  3       *day = tempDay;

646  3       /* If there is more than one backup, use the latest backup */
647  3       if (backupTimesCount[tempDay] > 1)
648  4       {
649  4         CBOX_GoId (backupTimeBox[tempDay], 0);
650  4         CBOX_CurSelect (backupTimeBox[tempDay]);
651  3       }

653  3       /* Else use the only backup on this day */
654  3       else
655  4       {
656  4         currentSelectedTime = backupTimes[tempDay][0];
657  3       }
658  3       found = BOOL_TRUE;
659  2     }
660  2     else
661  3     {
662  3       /* Go to the previous day */
663  3       tempDay--;
664  2     }
665  1   }

667  1   /* Return wheter or not we found it */
668  1   return (found);
669   }
```

```
671  /*********************************************************************
672   * REST_CalendarSetPrevBackup
673   *
674   * Description:
675   *   This routine will select the previous backup (
                                                    chronologically) to the
676   *   currently selected backup.
677   *
678   * Parameters:
679   *   None.
680   *
681   * Returns:
682   *   BOOL_TRUE  - If a previous backup was found and selected
683   *   BOOL_FALSE - otherwise
684   *
685   *********************************************************************/

687  BoolEnum REST_CalendarSetPrevBackup (void)
688 1 {
689 1   BoolEnum found = BOOL_FALSE;
                                /* Flag if we found a previous backup */
690 1   BoolEnum status;
                                 /* Flag if previous exists for time */
691 1   Int      foundDay;             /* Day of previous backup */
692 1   Int      tempDay;              /* Temporary day holder */
693 1   Int      tempMonth;            /* Temporary month holder */
694 1   Int      index;                /* Loop index */
695 1   BoolEnum returnStatus = BOOL_FALSE;/* Status to return */
696 1   u_long   flags;

698 1   if (TBUT_GetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton))
699 1     flags = BACKUP_SELECTION_FLAG_PARTIAL_OK;
700 1   else
701 1     flags = BACKUP_SELECTION_FLAG_COMPLETE_ONLY;

703 1   /*
704 1    * Find the previous backup time
705 1    */

707 1   /* If we are in the selected month/year,
                                       move from the current selection */
708 1   if ((displayedMonth == selectedMonth) &&
709 1       (displayedYear == selectedYear))
710 2   {

712 2     /* If this is the only backup or the earliest backup of the day */
713 2     index = (Int)CBOX_ChosenGetId (backupTimeBox[selectedDay]);
714 2     if ((backupTimesCount[selectedDay] == 1) ||
715 2         (index == backupTimesCount[selectedDay] - 1))
716 3     {

718 3       /* Find the previous day in the month with a backup */
719 3       foundDay = selectedDay;
720 3       found = REST_SetPreviousBackupInMonth (&foundDay);
721 2     }
722 2     else
723 3     {
724 3       /* There is an earlier backup on this day */
725 3       foundDay = selectedDay;
726 3       CBOX_GoId (backupTimeBox[selectedDay], index+1);
727 3       CBOX_CurSelect (backupTimeBox[selectedDay]);
728 3       found = BOOL_TRUE;
729 2     }
730 1   }
```



```
732 1   /* If we haven't found it yet, try the prev month */
733 1   if (!found)
734 2   {
735 2     /* First, check if a previous backup exists */
736 2     EDMRST_IsTherePrevBackupForTime (GREST_Handle,
737 2                                      currentSelectedTime,
738 2                                      flags,
739 2                                      &status);
740 2     if (status)
741 3     {

743 3       /* Find the previous month with a backup (we know one exists) */
744 3       tempMonth = displayedMonth - 1;
745 3       while (!found)
746 4       {

748 4         /* Only consider this month if backups were done in it */
749 4         if (REST_BackupExistsInMonth (tempMonth, displayedYear))
750 5         {
751 5           displayedMonth--;
752 5           REST_UpdateDisplayedDate();

754 5           /* get the last day of the month (
                                          plus one so we can find the last) */
755 5           foundDay = GTIME_DaysPerMonth (
                                             displayedMonth, displayedYear);
756 5           found = REST_SetPreviousBackupInMonth (&foundDay);
757 4         }
758 4         tempMonth--;

760 3       }
761 2     }
762 1   }

764 1   /* If we found one */
765 1   if (found)
766 2   {

768 2     /* Redraw the old selected day (has affect of unselecting) */
769 2     REST_DrawSelectedDay (BOOL_FALSE);

771 2     /* Set the selecte backup date */
772 2     selectedDay = foundDay;
773 2     selectedMonth = displayedMonth;
774 2     selectedYear = displayedYear;

776 2     /* Draw the newly selected day */
777 2     REST_DrawSelectedDay (BOOL_TRUE);

779 2     returnStatus = BOOL_TRUE;
780 1   }

782 1   return (returnStatus);
783   }
```

```
/*****************************************************************
 * REST_SetNextBackupInMonth
 *
 * Description:
 *   This routine will select the next backup (chronologically) done
 *   after the given day of the current month and year.
 *
 * Parameters:
 *   day (IO) - the day to start at, updated to found day (if any)
 *
 * Returns:
 *   BOOL_TRUE  - If a next backup was found and selected
 *   BOOL_FALSE - otherwise
 *
 *****************************************************************/

static BoolEnum REST_SetNextBackupInMonth (Int *day)
{
  BoolEnum found = BOOL_FALSE;
                                   /* Flag if we found a next backup */
  Int      tempDay;                /* Temporary day holder */
  Int      index;                  /* Loop index */
  Int      numberDays;             /* Number of days in the month */
  BoolEnum returnStatus = BOOL_FALSE;/* Status to return */

  /* Find the next day with a backup */
  tempDay = *day + 1;
  numberDays = GTIME_DaysPerMonth (displayedMonth, displayedYear);
  while ((!found) && (tempDay < numberDays))
  {
    if (backupTimesCount[tempDay] > 0)
    {
      /* This is the day */
      *day = tempDay;

      /* If there is more than one backup, use the first backup */
      if (backupTimesCount[tempDay] > 1)
      {
        CBOX_GoId (
              backupTimeBox[tempDay], backupTimesCount[tempDay] - 1);
        CBOX_CurSelect (backupTimeBox[tempDay]);
      }

      /* Else use the only backup on this day */
      else
      {
        currentSelectedTime = backupTimes[tempDay][0];
      }
      found = BOOL_TRUE;
    }
    else
    {
      /* Go to the next day */
      tempDay++;
    }
  }

  /* Return wheter or not we found it */
  return (found);
}
```



```
/*****************************************************************
 * REST_CalendarSetNextBackup
 *
 * Description:
 *   This routine will select the next backup (chronologically) to the
 *   currently selected backup.
 *
 * Parameters:
 *   None.
 *
 * Returns:
 *   BOOL_TRUE  - If a next backup was found and selected
 *   BOOL_FALSE - otherwise
 *
 *****************************************************************/

BoolEnum REST_CalendarSetNextBackup (void)
{
  BoolEnum found = BOOL_FALSE;
                                   /* Flag if we found a previous backup */
  BoolEnum status;
                                  /* Flag if next exists for time */
  Int      foundDay;               /* Day of previous backup */
  Int      tempDay;                /* Temporary day holder */
  Int      tempMonth;              /* Temporary month holder */
  Int      index;                  /* Loop index */
  Int      numberDays;             /* Number of days in the month */
  BoolEnum returnStatus = BOOL_FALSE;/* Status to return */
  u_long    flags;

  if (TBUT_GetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton))
    flags = BACKUP_SELECTION_FLAG_PARTIAL_OK;
  else
    flags = BACKUP_SELECTION_FLAG_COMPLETE_ONLY;

  /*
   * Find the next backup time
   */

  /* If we are in the selected month/year,
                                        move from the current selection */
  if ((displayedMonth == selectedMonth) &&
      (displayedYear == selectedYear))
  {
    /* If this is the only backup or the last backup of the day */
    index = (Int)CBOX_ChosenGetId (backupTimeBox[selectedDay]);
    if ((backupTimesCount[selectedDay] == 1) ||
        (CBOX_ChosenGetId (backupTimeBox[selectedDay]) == 0))
    {

      /* Find the next day in the month with a backup */
      foundDay = selectedDay;
      found = REST_SetNextBackupInMonth (&foundDay);
    }
    else
    {
      /* There is a later backup on this day */
      index = (Int) CBOX_ChosenGetId (backupTimeBox[selectedDay]);
      foundDay = selectedDay;
      CBOX_GoId (backupTimeBox[selectedDay], index-1);
      CBOX_CurSelect (backupTimeBox[selectedDay]);
      found = BOOL_TRUE;
    }
  }
```

```
906 1   /* If we haven't found it yet, try the next month */
907 1   if (!found)
908 2   {

910 2     /* First, check if a previous backup exists */
911 2     EDMRST_IsThereNextBackupForTime (GREST_Handle,
912 2                                      currentSelectedTime,
913 2                                      flags,
914 2                                      &status);
915 2     if (status)
916 3     {

918 3       /* Find the previous next with a backup (we know one exists) */
919 3       tempMonth = displayedMonth + 1;
920 3       while (!found)
921 4       {

923 4         /* Only consider this month if backups were done in it */
924 4         if (REST_BackupExistsInMonth (tempMonth, displayedYear))
925 5         {
926 5           displayedMonth++;
927 5           REST_UpdateDisplayedDate();

929 5           /* use -1, day 0 should be checked */
930 5           foundDay = -1;
931 5           found = REST_SetNextBackupInMonth (&foundDay);
932 4         }
933 4         tempMonth++;
934 3       }
935 2     }
936 1   }

938 1   /* If we found one */
939 1   if (found)
940 2   {

942 2     /* Redraw the old selected day (has affect of unselecting) */
943 2     REST_DrawSelectedDay (BOOL_FALSE);

945 2     /* Set the selecte backup date */
946 2     selectedDay = foundDay;
947 2     selectedMonth = displayedMonth;
948 2     selectedYear = displayedYear;

950 2     /* Draw the newly selected day */
951 2     REST_DrawSelectedDay (BOOL_TRUE);

953 2     returnStatus = BOOL_TRUE;
954 1   }

956 1   return (returnStatus);
957   }
```



```
959   /*************************************************************************
960    * REST_CalendarDraw
961    *
962    * Description:
963    *   This routine will draw the calendar.
964    *
965    * Parameters:
966    *   None.
967    *
968    * Returns:
969    *   None.
970    *
971    *************************************************************************/

973   void REST_CalendarDraw (void)
974 1 {
975 1   Int           x;                          /* x location for drawing */
976 1   Int           y;                          /* y location for drawing */
977 1   Point16Rec    startPoint;                 /* Start point for line */
978 1   Point16Rec    endPoint;                   /* End point for line */
979 1   int           dayNumber;                  /* Day number to draw */
980 1   Rect16Rec     textRect;                   /* Bounding rectangle for text */
981 1   Point16Rec    textExt;                    /* Extents needed for text */
982 1   Char          dayString[SMALL_STRING_LENGTH]; /* String to draw */
983 1   Int           dayWidth;                   /* The width of a day box */
984 1   Int           dayHeight;                  /* The height of a day box */
985 1   int           numberDays;                 /* Number of days in the displayed month */
986 1   int           numberWeeks;                /* Number of weeks in the displayed month */

988 1   /* Get the width and height of the day */
989 1   REST_GetDayWidthHeight (&dayWidth, &dayHeight);

991 1   DRAW_SetColors ((WgtPtr)REST_CalendarWindow->CalendarPanel,
992 1                   COLOR_Black(),
993 1                   COLOR_Transparent());
994 1   DRAW_SetPen((WgtPtr)REST_CalendarWindow->CalendarPanel, PEN_Solid());

996 1   /* Draw the top line over the day labels */
997 1   startPoint.x = MARGIN_WIDTH;
998 1   startPoint.y = 0;
999 1   endPoint.x = startPoint.x + (dayWidth * DAYS_PER_WEEK);
1000 1  endPoint.y = startPoint.y;
1001 1  DRAW_Line ((
       WgtPtr)REST_CalendarWindow->CalendarPanel, &startPoint, &endPoint);

1003 1  /* Draw the day labels */
1004 1  DRAW_SetFont ((
              WgtPtr)REST_CalendarWindow->CalendarPanel, FONT_Normal());
1005 1  for (x = 0; x<DAYS_PER_WEEK; x++)
1006 2  {
1007 2    STR_Sprintf (dayString, "%s", GTIME_GetDayStr(x+1));
1008 2    textRect.Ori.x = MARGIN_WIDTH + (x * dayWidth) + DAY_LABEL_OFFSET;
1009 2    textRect.Ori.y = 0;
1010 2    textRect.Ext.x = dayWidth;
```

```
1011  2      textRect.Ext.y = DAY_LABEL_HEIGHT;
1012  2      DRAW_Text ((WgtPtr)REST_CalendarWindow->CalendarPanel,
1013  2                 &textRect,
1014  2                 DRAW_JUSTCENTER,
1015  2                 dayString);
1016  1    }

1018  1    /* Determine the number of days and weeks in the displayed month */
1019  1    numberDays = GTIME_DaysPerMonth (displayedMonth, displayedYear);
1020  1    numberWeeks = GTIME_NumberWeeks (
                                displayedMonthTime.tm_wday, numberDays);

1022  1    /* Draw the vertical lines */
1023  1    for (x = 0; x<=DAYS_PER_WEEK; x++)
1024  2    {
1025  2      startPoint.x = MARGIN_WIDTH + (x * dayWidth);
1026  2      startPoint.y = 0;

1028  2      endPoint.x = startPoint.x;
1029  2      endPoint.y = DAY_LABEL_HEIGHT + (numberWeeks * dayHeight);
1030  2      DRAW_Line ((
       WgtPtr)REST_CalendarWindow->CalendarPanel, &startPoint, &endPoint);
1031  1    }

1033  1    /* Draw the horizontal lines */
1034  1    for (y = 0; y<=numberWeeks; y++)
1035  2    {
1036  2      startPoint.x = MARGIN_WIDTH;
1037  2      startPoint.y = DAY_LABEL_HEIGHT + (y * dayHeight);

1039  2      endPoint.x = MARGIN_WIDTH + (dayWidth * DAYS_PER_WEEK);
1040  2      endPoint.y = startPoint.y;
1041  2      DRAW_Line ((
       WgtPtr)REST_CalendarWindow->CalendarPanel, &startPoint, &endPoint);
1042  1    }

1044  1    /* Draw the date labels */
1045  1    dayNumber = displayedMonthTime.tm_wday;
1046  1    textRect.Ori.x = MARGIN_WIDTH + (
                                dayNumber * dayWidth) + DAY_LABEL_OFFSET;
1047  1    textRect.Ori.y = DAY_LABEL_HEIGHT + DAY_LABEL_OFFSET;
1048  1    for (x = 1; x <= numberDays; x++)
1049  2    {
1050  2      STR_Sprintf (dayString, "%d", x);
1051  2      DRAW_SetFont ((
                WgtPtr)REST_CalendarWindow->CalendarPanel, FONT_Normal());
1052  2      DRAW_QueryTextExt((
          WgtPtr)REST_CalendarWindow->CalendarPanel, dayString, &textExt);
1053  2      textRect.Ext.x = textExt.x;
1054  2      textRect.Ext.y = textExt.y;

1056  2      DRAW_SetColors ((WgtPtr)REST_CalendarWindow->CalendarPanel,
1057  2                      COLOR_Black(),
1058  2                      COLOR_Transparent());
1059  2      DRAW_SetPen ((WgtPtr)REST_CalendarWindow->CalendarPanel, PEN_Solid(
                                                                      ));
1060  2      DRAW_Text ((WgtPtr)REST_CalendarWindow->CalendarPanel,
1061  2                 &textRect,
1062  2                 DRAW_JUSTCENTER,
1063  2                 dayString);

1065  2      /* Set up the next day */
1066  2      dayNumber++;

1068  2      /* If we have hit the end of the week */
1069  2      if (dayNumber == DAYS_PER_WEEK)
```



```
1070  3      {
1071  3        /* Set the x coords back to the start */
1072  3        dayNumber = 0;
1073  3        textRect.Ori.x = MARGIN_WIDTH + DAY_LABEL_OFFSET;

1075  3        /* Bump the y coords the height of a day */
1076  3        textRect.Ori.y += dayHeight;
1077  2      }
1078  2      else
1079  3      {
1080  3        /* Just bump the x coords the width of a day */
1081  3        textRect.Ori.x += dayWidth;
1082  2      }
1083  1    }

1085  1    /* Draw the selected Day */
1086  1    REST_DrawSelectedDay (BOOL_TRUE);

1088    }
```

```
1090  /*****************************************************************
1091   * REST_CalendarResize
1092   *
1093   * Description:
1094   *   This routine handles the resizing of the calendar.  It will then
1095   *   resize and relocate all combo boxes.
1096   *
1097   * Parameters:
1098   *   None.
1099   *
1100   * Returns:
1101   *   None.
1102   *
1103   *****************************************************************/

1105  void REST_CalendarResize (void)
1106 1 {
1107 1   Int           index;        /* loop counter */
1108 1   int           dayNumber;
                        /* week day number for determining x position */
1109 1   Rect16Rec     textRect;     /* Rectangle to draw the text in */
1110 1   Point16Rec    textExt;      /* Extents necessary for the text */
1111 1   Char          dayString[SMALL_STRING_LENGTH];
                                    /* temporary string for test */
1112 1   Int           dayWidth;     /* The width of a day box */
1113 1   Int           dayHeight;    /* The height of a day box */
1114 1   int           numberDays;
                              /* Number of days in the displayed month */

1116 1   /* If we haven't initialized the boxes yet, get out */
1117 1   if (boxesInited != BOOL_TRUE)
1118 2   {
1119 2     return;
1120 1   }

1122 1   /* Get the width and height of the day */
1123 1   REST_GetDayWidthHeight (&dayWidth, &dayHeight);

1125 1   /* Determine the position for each of the choice boxes */
1126 1   /* Get a sample string to determin string height */
1127 1   DRAW_SetFont ((
              WgtPtr)REST_CalendarWindow->CalendarPanel, FONT_Normal());
1128 1   STR_Cpy (dayString, "88");
1129 1   DRAW_QueryTextExt((
          WgtPtr)REST_CalendarWindow->CalendarPanel, dayString, &textExt);

1131 1   dayNumber = displayedMonthTime.tm_wday;
1132 1   textRect.Ori.x = MARGIN_WIDTH + dayNumber * dayWidth;
1133 1   textRect.Ori.y = DAY_LABEL_HEIGHT + DAY_LABEL_OFFSET + textExt.y;
1134 1   textRect.Ext.x = dayWidth;
1135 1   textRect.Ext.y = dayHeight - textExt.y - DAY_LABEL_OFFSET;

1137 1   /* Determine the number of days in the displayed month */
1138 1   numberDays = GTIME_DaysPerMonth (displayedMonth, displayedYear);

1140 1   /* Loop through all days and set up the box positions for that day
                                                                      */
1141 1   for (index = 0; index < numberDays; index++)
1142 2   {

1144 2     /* Position the time CBox and Text widget in the middle of the day
                                                                      */
1145 2     REST_CalSetBoxPos (index, textRect.Ori, textRect.Ext);

1147 2     /* Set up the next day */
```



```
1148 2     dayNumber++;

1150 2     /* If we have hit the end of the week */
1151 2     if (dayNumber == DAYS_PER_WEEK)
1152 3     {
1153 3       /* Set the x coords back to the start */
1154 3       dayNumber = 0;
1155 3       textRect.Ori.x = MARGIN_WIDTH;

1157 3       /* Bump the y coords the height of a day */
1158 3       textRect.Ori.y += dayHeight;
1159 2     }
1160 2     else
1161 3     {
1162 3       /* Just bump the x coords the width of a day */
1163 3       textRect.Ori.x += dayWidth;
1164 2     }
1165 1   }

1167    }
```

```
1169  /*****************************************************************
1170   * REST_CalSetBoxSelected
1171   *
1172   * Description:
1173   *   This routine will select the day and time for the given combo
                                                                    boxes
1174   *   currently selected time.
1175   *
1176   * Parameters:
1177   *   cbox (I) - The combo box which was selected
1178   *
1179   * Returns:
1180   *   None.
1181   *
1182   *****************************************************************/

1184  static void REST_CalSetBoxSelected (CBoxPtr cbox)
1185 1{
1186 1  Int  newId; /* The new selection Id */

1188 1  /* Redraw the old selected day, unselected */
1189 1  REST_DrawSelectedDay (BOOL_FALSE);

1191 1  /* Get the new selected time */
1192 1  selectedDay = (Int) WGT_GetClientRes ((WgtPtr)cbox);
1193 1  selectedMonth = displayedMonth;
1194 1  selectedYear = displayedYear;
1195 1  newId = (Int) CBOX_ChosenGetId (cbox);
1196 1  currentSelectedTime = backupTimes[selectedDay][newId];

1198 1  /* Draw the newly selected day */
1199 1  REST_DrawSelectedDay (BOOL_TRUE);

1201  }
```



```
1203  /*****************************************************************
1204   * REST_SetDisplayedDate
1205   *
1206   * Description:
1207   *   This routine will display the current month and year in the date
1208   *   text area.
1209   *
1210   * Parameters:
1211   *   None.
1212   *
1213   * Returns:
1214   *   None.
1215   *
1216   *****************************************************************/

1218  static void REST_SetDisplayedDate (void)
1219 1{
1220 1  Char  displayString[MEDIUM_STRING_LENGTH];  /* String to display */

1222 1  /* Set the month display string */
1223 1  STR_Sprintf (displayString, "%s", GTIME_GetMonthStr (
                                                        displayedMonth));
1224 1  TAREA_SetLabel (REST_CalendarWindow->MonthTArea, displayString);

1226 1  /* Set the year display string */
1227 1  STR_Sprintf (displayString, "%d", 1900 + displayedYear);
1228 1  TAREA_SetLabel (REST_CalendarWindow->YearTArea, displayString);
1229  }
```

```
1231  /*****************************************************************
1232   * REST_UpdateDisplayedDate
1233   *
1234   * Description:
1235   *   This routine will update the displayed date to the current
                                                              displayed
1236   *   month and year.  It will redraw the calendar showing the backups
1237   *   for each day of the current month.
1238   *
1239   * Parameters:
1240   *   None.
1241   *
1242   * Returns:
1243   *   None.
1244   *
1245   *****************************************************************/

1247  static void REST_UpdateDisplayedDate (void)
1248 1 {
1249 1   int         startDay = 1;
                                  /* Start day to get start month time */
1250 1   time_t      endMonthTime;     /* Last time for the month */
1251 1   time_t      startMonthTime;   /* First time for the month */
1252 1   struct tm *nextTime;
                                  /* Next time to examine in all times */
1253 1   struct tm   thisTime;         /* Temporary time storage */
1254 1   long        cookie = INIT_COOKIE;/* Ah, the magic cookie */
1255 1   short       numberEntries;    /* Number of times returned */
1256 1   time_t     *times;            /* Times returned from API */
1257 1   eerrno_ty   eerrno;           /* Returned error code */
1258 1   int         monthDay;         /* Day of month index from time */
1259 1   Int         i;                /* Loop counter */
1260 1   Char        timeString[MEDIUM_STRING_LENGTH];  /* Converted string */
1261 1   BoolEnum    first;
                                /* Flag if this is the first time for day */
1262 1   u_long      flags;

1264 1   if (TBUT_GetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton))
1265 1     flags = BACKUP_SELECTION_FLAG_PARTIAL_OK;
1266 1   else
1267 1     flags = BACKUP_SELECTION_FLAG_COMPLETE_ONLY;

1269 1   /* Update the current date */
1270 1   GTIME_UpdateDate (&startDay,
1271 1                     &displayedMonth,
1272 1                     &displayedYear,
1273 1                     BOOL_FALSE);
1274 1   REST_SetDisplayedDate ();

1276 1   /* Get the start time for the month */
1277 1   GTIME_GetTimeTm (startDay,
1278 1                    displayedMonth,
1279 1                    displayedYear,
1280 1                    0,
1281 1                    0,
1282 1                    &displayedMonthTime);
1283 1   startMonthTime = mktime (&displayedMonthTime);

1285 1   /* Get the end time for the month */
1286 1   GTIME_GetTimeTm (GTIME_DaysPerMonth(displayedMonth, displayedYear),
1287 1                    displayedMonth,
1288 1                    displayedYear,
1289 1                    HOURS_PER_DAY - 1,
1290 1                    MINUTES_PER_HOUR - 1,
1291 1                    &thisTime);
```



```
1292 1   endMonthTime = mktime (&thisTime);

1294 1   /* Re-initialize the counts, and the cboxes */
1295 1   for (i=0; i<MAX_DAYS_PER_MONTH; i++)
1296 2   {
1297 2     backupTimesCount[i] = 0;
1298 2     CBOX_GoFirst(backupTimeBox[i]);
1299 2     while (CBOX_IsOk (backupTimeBox[i]))
1300 2       CBOX_CurRemoveElt (backupTimeBox[i]);
1301 2     WGT_SetInvisible ((WgtPtr)backupTimeBox[i]);
1302 2     WGT_SetInvisible ((WgtPtr)backupTimeText[i]);
1303 1   }

1305 1   /* Flag that we are selecting CBox values by code to stop CB
                                                              processing */
1306 1   selectionByCode = BOOL_TRUE;

1308 1   /* Allocate space for the times array */
1309 1   times = (time_t *) GUTIL_Malloc (TIME_BUFFER_LENGTH * sizeof(
                                                              time_t));


1311 1   /* Until the API says we're done, keep looping getting all times */
1312 1   while (cookie != DONE_COOKIE)
1313 2   {
1314 2     /* Get the next batch of times */
1315 2     if ((eerrno = EDMRST_GetAllBackupTimes (GREST_Handle,
1316 2                                             startMonthTime,
1317 2                                             endMonthTime,
1318 2                                             flags,
1319 2                                             TIME_BUFFER_LENGTH,
1320 2                                             times,
1321 2                                             &numberEntries,
1322 2                                             &cookie)) == E_SUCCESS)
1323 3     {

1325 3       /* Loop through all the backup times */
1326 3       for (i = 0; i < numberEntries; i++)
1327 4       {

1329 4         /* Get the time struct so we know the day of the month */
1330 4         nextTime = localtime (&times[i]);

1332 4         /* Get the day of the month index into the arrays */
1333 4         monthDay = nextTime->tm_mday - 1;

1335 4         /* Make sure we have space for this time */
1336 4         if (backupTimesCount[monthDay] < MAX_TIME_SLOTS)
1337 5         {

1339 5           /* Store this time in the right place in the array of times
                                                                  */
1340 5           backupTimes[monthDay][backupTimesCount[monthDay]] = times[i];

1342 5           /* Get the time in a displayable format */
1343 5           strftime (timeString,
1344 5                     MEDIUM_STRING_LENGTH,
1345 5                     "%H:%M",
1346 5                     localtime(&times[i]));

1348 5           /* Since we have a time for the date,
                                                 make the choice box visible */
1349 5           if (backupTimesCount[monthDay] == 0)
1350 6           {
1351 6             WGT_SetVisible ((WgtPtr)backupTimeText[monthDay]);
1352 6             TED_SetStr (backupTimeText[monthDay], timeString);
1353 5           }
```

```
1354  5            else
1355  6            {
1356  6              WGT_SetInvisible ((WgtPtr)backupTimeText[monthDay]);
1357  6              WGT_SetVisible ((WgtPtr)backupTimeBox[monthDay]);
1358  5            }

1360  5            /*
1361  5             * NOTE:
1362  5             *
1363  5             *  The times are given to us in descending order.
                                                                  We want to
1364  5             *  put them in the list such that the first is the earliest
                                                                       and
1365  5             *  the last is the latest.
                                                 If we go to the ID of the last one
1366  5             *  we got and do a CBOX_CurAddElt it will put it before it
                                                                       so
1367  5             *  that will reverse the order for us.
                                                       The latest backup on
1368  5             *  that day will then be 0, and the earliest will then be
1369  5             *  backupTimesCount[date].
1370  5             */

1372  5            /* Get to the end of the list */
1373  5            if (backupTimesCount[monthDay] > 0)
1374  6            {
1375  6              first = BOOL_FALSE;
1376  6              CBOX_GoId (backupTimeBox[monthDay],
1377  6                         backupTimesCount[monthDay]);
1378  5            }
1379  5            else
1380  6            {
1381  6              first = BOOL_TRUE;
1382  6              CBOX_GoFirst(backupTimeBox[monthDay]);
1383  5            }

1385  5            /* Add this time to the element list */
1386  5            CBOX_CurAddElt (backupTimeBox[monthDay], NULL);

1388  5            /* Set the label for this element to the time string */
1389  5            CBOX_CurSetLabel (backupTimeBox[monthDay], timeString);

1391  5            /* Set the ID to the index into the backup times */
1392  5            CBOX_CurSetId (backupTimeBox[monthDay],
1393  5                           backupTimesCount[monthDay]);

1395  5            /* If this is the current time or the first for that day,
                                                                  select it */
1396  5            if ((times[i] == currentSelectedTime) || first)
1397  5              CBOX_CurSelect (backupTimeBox[monthDay]);

1399  5            /* Also select the day */
1400  5            if (times[i] == currentSelectedTime)
1401  5              REST_CalSetBoxSelected (backupTimeBox[monthDay]);

1403  5            /* Update the time count for this day */
1404  5            backupTimesCount[monthDay]++;
1405  4          }
1406  3        }

1408  2      }
1409  2      else
1410  3      {
1411  3        /* Nothing more to see here folks... Nothing more to see... */
1412  3        cookie = DONE_COOKIE;
1413  2      }
```



```
1414  1    }

1416  1    /* Free the times array, we be done with it */
1417  1    GUTIL_Free (times);

1419  1    /* Remove the selection by code flag */
1420  1    selectionByCode = BOOL_FALSE;

1422  1    /* Resize the window, for different number of weeks in this month */
1423  1    REST_CalendarResize ();

1425  1    /* Invalidate the calendar to force a redraw */
1426  1    WGT_Inval ((WgtPtr)REST_CalendarWindow->CalendarPanel, BOOL_TRUE);

1428    }
```

```
1430  /*****************************************************************
1431   * REST_CalendarGetMostRecentTime
1432   *
1433   * Description:
1434   *   This routine will determine the time of the most recent backup.
1435   *
1436   * Parameters:
1437   *   None.
1438   *
1439   * Returns:
1440   *   None.
1441   *
1442   *****************************************************************/

1444  time_t REST_CalendarGetMostRecentTime (void)
1445 1{
1446 1  time_t     mostRecentTime = 0;  /* Time of most recent backup */
1447 1  int        i;                   /* Loop index */
1448 1  long       cookie = INIT_COOKIE;/* Ah, the magic cookie */
1449 1  short      numberEntries;       /* Number of times returned */
1450 1  time_t    *times;               /* Array of times returned */
1451 1  eerrno_ty  eerrno;              /* Error code returned */
1452 1  u_long     flags;

1454 1  if (TBUT_GetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton))
1455 1    flags = BACKUP_SELECTION_FLAG_PARTIAL_OK;
1456 1  else
1457 1    flags = BACKUP_SELECTION_FLAG_COMPLETE_ONLY;

1459 1  /* Allocate space for the times */
1460 1  times = (time_t *) GUTIL_Malloc (TIME_BUFFER_LENGTH * sizeof(
                                                              time_t));


1462 1  /* Keep looping until no more times are found */
1463 1  while (cookie != DONE_COOKIE)
1464 2  {

1466 2    /* Get all times (
                we can start with the current selection to minimize) */
1467 2    if ((eerrno = EDMRST_GetAllBackupTimes (GREST_Handle,
1468 2                                            currentSelectedTime,
1469 2                                            time(NULL),
1470 2                                            flags,
1471 2                                            TIME_BUFFER_LENGTH,
1472 2                                            times,
1473 2                                            &numberEntries,
1474 2                                            &cookie)) == 0)
1475 3    {

1477 3      /* Loop through all the backup times */
1478 3      for (i = 0; i < numberEntries; i++)
1479 4      {

1481 4        /* Check if this is more recent then the current */
1482 4        if (times[i] > mostRecentTime)
1483 4          mostRecentTime = times[i];
1484 3      }

1486 2    }
1487 2    else
1488 3    {
1489 3      /* Nothing more to see here folks... Nothing more to see... */
1490 3      cookie = DONE_COOKIE;
1491 2    }
1492 1  }
```



```
1494 1  /* Free the times array */
1495 1  GUTIL_Free (times);

1497 1  return (mostRecentTime);
1498  }
```

```
1500  /*************************************************************************
1501   * REST_CalendarUpdateDate
1502   *
1503   * Description:
1504   *   This routine will handle the callback to update the displayed
1505   *   month based on the current selection in the Month and Year
1506   *   combo boxes.
1507   *
1508   * Parameters:
1509   *   newMonth      (I) - The new month number to display
1510   *   newYear       (I) - The new year number to display
1511   *   isMonthUpdate (I) - Flag if this update is a month update
1512   *
1513   * Returns:
1514   *   None.
1515   *
1516   *************************************************************************/

1518  static void REST_CalendarUpdateDate (int      newMonth,
1519                                       int      newYear,
1520                                       BoolEnum isMonthUpdate)
1521 1 {
1522 1   int       startDay = 1;
                              /* Temporary storage for first day of month */
1523 1   int       tempMonth;     /* Temporary month storage for new month */
1524 1   int       tempYear;      /* Temporary year storage */
1525 1   struct tm thisTime;
                            /* Time struct for last time in previous month */
1526 1   time_t    endMonthTime;  /* Last time in previous month */
1527 1   BoolEnum  previous;
                             /* Flag if the selected is prior to current */
1528 1   int       status = 0;    /* Return status from GREST calls */
1529 1   boolean_ty isThere = 0;  /* Flag if the backup exists */
1530 1   int       currentDay;    /* Today's day */
1531 1   int       currentMonth;  /* Today's month */
1532 1   int       currentYear;   /* Today's year */
1533 1   Char      outputString[MAX_STRING_LENGTH];
                                          /* Error string to display */
1534 1   time_t    mostRecentTime = 0;  /* Time of most recent backup */
1535 1   int       tmpDay;        /* Temporary storage */
1536 1   int       tmpHour;       /* Temporary storage */
1537 1   int       tmpMinutes;    /* Temporary storage */
1538 1   u_long    flags;
1539 1   BoolEnum  dateFault = BOOL_FALSE;
                             /* Flag if the date could not be changed */


1541 1   if (TBUT_GetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton))
1542 1     flags = BACKUP_SELECTION_FLAG_PARTIAL_OK;
1543 1   else
1544 1     flags = BACKUP_SELECTION_FLAG_COMPLETE_ONLY;

1546 1   /* Get the adjusted month and year */
1547 1   tempMonth = newMonth;
1548 1   tempYear = newYear;
1549 1   GTIME_UpdateDate (&startDay,
1550 1                     &tempMonth,
1551 1                     &tempYear,
1552 1                     BOOL_FALSE);

1554 1   /*
1555 1    * Determine if the user is going past the current day, don't allow
1556 1    * it since there can't be any future backups (I hope)
1557 1    */

1559 1   /* Get the current date */
```



```
1560 1   GTIME_GetCurrentDate (&currentDay, &currentMonth, &currentYear);
1561 1   if ((tempYear > currentYear) ||
1562 1       ((tempYear == currentYear) && (tempMonth > currentMonth)))
1563 2   {
1564 2     /*
1565 2      * If this is not a month update, determine the most recent month
1566 2      * and go there
1567 2      */

1569 2     if (!isMonthUpdate)
1570 3     {
1571 3       /* Display the error message */
1572 3       GALERT_DisplayError ((WinPtr)REST_CalendarWindow,
1573 3                            REST_GetErrorString (REST_WARNING_INDEX),
1574 3                            GICON_GetWarning(),
1575 3                            REST_GetErrorString (
                                                   REST_NO_FUTURE_MESSAGE));
1576 3       mostRecentTime = REST_CalendarGetMostRecentTime ();

1578 3       /* If there is a more recent time */
1579 3       if (mostRecentTime != 0)
1580 4       {
1581 4         /* Get the components of the time */
1582 4         GTIME_BreakDownTime (mostRecentTime,
1583 4                              &tmpDay,
1584 4                              &displayedMonth,
1585 4                              &displayedYear,
1586 4                              &tmpHour,
1587 4                              &tmpMinutes);
1588 4         REST_UpdateDisplayedDate();
1589 3       }

1591 3       dateFault = BOOL_TRUE;
1592 2     }
1593 1   }

1595 1   /* Determine if this is a previous or next operation */
1596 1   if ((tempYear < displayedYear) ||
1597 1       ((tempYear == displayedYear) && (tempMonth < displayedMonth)))
1598 1     previous = BOOL_TRUE;
1599 1   else
1600 1     previous = BOOL_FALSE;

1602 1   /* Get the last time in the month */
1603 1   GTIME_GetTimeTm (GTIME_DaysPerMonth(tempMonth, tempYear),
1604 1                    tempMonth,
1605 1                    tempYear,
1606 1                    HOURS_PER_DAY - 1,
1607 1                    MINUTES_PER_HOUR - 1,
1608 1                    &thisTime);
1609 1   endMonthTime = mktime (&thisTime);

1611 1   /* If there are no backups in this or any previous/next month,
                                                          warn user */
1612 1   if (!dateFault && !REST_BackupExistsInMonth (tempMonth, tempYear))
1613 2   {
1614 2     if (previous)
1615 3     {
1616 3       status = EDMRST_IsTherePrevBackupForTime (GREST_Handle,
1617 3                                                 endMonthTime,
1618 3                                                 flags,
1619 3                                                 &isThere);
1620 3       if ((status == E_SUCCESS) && !isThere)
1621 4       {
1622 4         STR_Sprintf (outputString,
1623 4                      REST_GetErrorString (REST_NO_PREVIOUS_FORMAT),
```

```
1624  4                       GTIME_GetMonthStr(tempMonth),
1625  4                       1900 + tempYear);

1627  4          /* Display the error message */
1628  4          GALERT_DisplayError ((WinPtr)REST_CalendarWindow,
1629  4                               REST_GetErrorString (REST_WARNING_INDEX),
1630  4                               GICON_GetWarning(),
1631  4                               outputString);

1633  4          dateFault = BOOL_TRUE;
1634  3        }
1635  3        else if (status != E_SUCCESS)
1636  4        {
1637  4  /*
1638  4          printf (
                      "Error returned from EDMRST_IsTherePrevBackupForTime\n");
1639  4  */
1640  3        }
1641  2      }
1642  2      else
1643  3      {
1644  3        status = EDMRST_IsThereNextBackupForTime (GREST_Handle,
1645  3                                                  endMonthTime,
1646  3                                                  flags,
1647  3                                                  &isThere);
1648  3        if ((status == E_SUCCESS) && !isThere)
1649  4        {
1650  4          STR_Sprintf (outputString,
1651  4                       REST_GetErrorString (REST_NO_NEXT_FORMAT),
1652  4                       GTIME_GetMonthStr(tempMonth),
1653  4                       1900 + tempYear);

1655  4          /* Display the error message */
1656  4          GALERT_DisplayError ((WinPtr)REST_CalendarWindow,
1657  4                               REST_GetErrorString (REST_WARNING_INDEX),
1658  4                               GICON_GetWarning(),
1659  4                               outputString);

1661  4          dateFault = BOOL_TRUE;
1662  3        }
1663  3        else if (status != E_SUCCESS)
1664  4        {
1665  4  /*
1666  4          printf (
                      "Error returned from EDMRST_IsThereNextBackupForTime\n");
1667  4  */
1668  3        }
1669  2      }
1670  1    }

1672  1    if (!dateFault)
1673  2    {
1674  2      /* Update the displayed month and year */
1675  2      displayedMonth = tempMonth;
1676  2      displayedYear = tempYear;
1677  2      REST_UpdateDisplayedDate();
1678  1    }
1679       }
```



```
1681  /*********************************************************************
1682   * REST_CalendarPreviousMonth
1683   *
1684   * Description:
1685   *   This routine will handle the callback to update the displayed
1686   *   month to the previous month.
1687   *
1688   * Parameters:
1689   *   None.
1690   *
1691   * Returns:
1692   *   None.
1693   *
1694   *********************************************************************/

1696  void REST_CalendarPreviousMonth (void)
1697  1 {
1698  1   REST_CalendarUpdateDate (
                              displayedMonth - 1, displayedYear, BOOL_TRUE);
1699  }
```

```
1701  /*************************************************************************
1702   * REST_CalendarNextMonth
1703   *
1704   * Description:
1705   *   This routine will handle the callback to update the displayed
1706   *   month to the next month.
1707   *
1708   * Parameters:
1709   *   None.
1710   *
1711   * Returns:
1712   *   None.
1713   *
1714   *************************************************************************/

1716  void REST_CalendarNextMonth (void)
1717  1 {
1718  1   REST_CalendarUpdateDate (
                          displayedMonth + 1, displayedYear, BOOL_TRUE);
1719  }
```



```
1721  /*************************************************************************
1722   * REST_CalendarPreviousYear
1723   *
1724   * Description:
1725   *   This routine will handle the callback to update the displayed
1726   *   Year to the previous Year.
1727   *
1728   * Parameters:
1729   *   None.
1730   *
1731   * Returns:
1732   *   None.
1733   *
1734   *************************************************************************/

1736  void REST_CalendarPreviousYear (void)
1737  1 {
1738  1   REST_CalendarUpdateDate (
                          displayedMonth, displayedYear - 1, BOOL_FALSE);
1739  }
```

```
1741  /************************************************************************
1742   * REST_CalendarNextYear
1743   *
1744   * Description:
1745   *   This routine will handle the callback to update the displayed
1746   *   Year to the next Year.
1747   *
1748   * Parameters:
1749   *   None.
1750   *
1751   * Returns:
1752   *   None.
1753   *
1754   ************************************************************************/

1756  void REST_CalendarNextYear (void)
1757  1 {
1758  1   REST_CalendarUpdateDate (
                          displayedMonth, displayedYear + 1, BOOL_FALSE);
1759    }
```



```
1761  /************************************************************************
1762   * REST_CalendarToday
1763   *
1764   * Description:
1765   *   This routine will handle the most recent button callback, it will
1766   *   display the month with the most recent backup and will select the
1767   *   most recent backup.
1768   *
1769   * Parameters:
1770   *   None.
1771   *
1772   * Returns:
1773   *   None.
1774   *
1775   ************************************************************************/

1777  void REST_CalendarToday (void)
1778  1 {
1779  1   time_t      mostRecentTime = 0;  /* Time of most recent backup */
1780  1   int         selectedHour;        /* Temporary storage */
1781  1   int         selectedMinutes;     /* Temporary storage */

1783  1   mostRecentTime = REST_CalendarGetMostRecentTime ();

1785  1   /* If there is a more recent time */
1786  1   if (mostRecentTime != 0)
1787  2   {

1789  2     /* Redraw the old selected day (has affect of unselecting) */
1790  2     REST_DrawSelectedDay (BOOL_FALSE);

1792  2     /* Set the selected time to the most recent backup */
1793  2     currentSelectedTime = mostRecentTime;

1795  2     /* Get the components of the time */
1796  2     GTIME_BreakDownTime (currentSelectedTime,
1797  2                          &selectedDay,
1798  2                          &selectedMonth,
1799  2                          &selectedYear,
1800  2                          &selectedHour,
1801  2                          &selectedMinutes);

1803  2     /* Display the most recent date */
1804  2     displayedMonth = selectedMonth;
1805  2     displayedYear = selectedYear;
1806  2     REST_UpdateDisplayedDate();


1809  2     /* Draw the newly selected day */
1810  2     REST_DrawSelectedDay (BOOL_TRUE);
1811  1   }

1813    }
```

```
1815  /*************************************************************************
1816   * REST_CalendarCancel
1817   *
1818   * Description:
1819   *   This routine will cancel the restore calendar window.  It will
1820   *   set the selected time to time 0, indicating the cancel action.
1821   *
1822   * Parameters:
1823   *   None.
1824   *
1825   * Returns:
1826   *   BOOL_TRUE  - If the cancel was effective
1827   *   BOOL_FALSE - If the cancel was already in place
1828   *
1829   *************************************************************************/

1831  BoolEnum REST_CalendarCancel (void)
1832  1 {

1834  1   /* If we haven't terminated yet, cancel */
1835  1   if (!REST_CalTerminated)
1836  2   {
1837  2     currentSelectedTime = 0;
1838  2     WIN_ModalReturn((WinPtr)REST_CalendarWindow, (ClientPtr)NULL);
1839  1   }


1842  1   return (!REST_CalTerminated);
1843    }
```



```
1845  /*************************************************************************
1846   * REST_CalendarOK
1847   *
1848   * Description:
1849   *   This routine will clost the restore calendar window with a new
1850   *   selected time.
1851   *
1852   * Parameters:
1853   *   None.
1854   *
1855   * Returns:
1856   *   None.
1857   *
1858   *************************************************************************/

1860  void REST_CalendarOK (void)
1861  1 {

1863  1   WIN_ModalReturn((WinPtr)REST_CalendarWindow, (ClientPtr)NULL);

1865    }
```

```
1867  /*****************************************************************
1868   * REST_DateBoxNfy
1869   *
1870   * Description:
1871   *   This routine handles all notifications for the combo boxes.
1872   *
1873   * Parameters:
1874   *   cbox (I) - The combo box receiving the notification.
1875   *   code (I) - The notification code received.
1876   *
1877   * Returns:
1878   *   None.
1879   *
1880   *****************************************************************/

1882  static  void  REST_DateBoxNfy (CBoxPtr     cbox,
1883                                 CBoxNfyEnum code)
1884  1 {
1885  1   Rect16Rec  box;  /* Widget box for the cbox */

1887  2   switch (code) {
1888  2   case CBOX_NFYELTSELECTED:
1889  2     CBOX_DefNfy(cbox, code);
1890  2     /* USER CODE */
1891  2     if (!selectionByCode)
1892  2       REST_CalSetBoxSelected (cbox);
1893  2     break;
1894  2   case CBOX_NFYGAINFOCUS:
1895  2   case CBOX_NFYFOCUSLOST:
1896  2     break;
1897  2   case CBOX_NFYINIT:
1898  2     CBOX_DefNfy(cbox, code);
1899  2     box.Ori.x = 0;
1900  2     box.Ori.y = 0;
1901  2     box.Ext.x = DATE_CBOX_WIDTH;
1902  2     box.Ext.y = DATE_CBOX_HEIGHT;
1903  2     WGT_Configure ((WgtPtr)cbox, &box.Ori, &box.Ext);
1904  2     break;
1905  2   default:
1906  2     CBOX_DefNfy(cbox, code);
1907  1   }

1909    }
```



```
1911  /*****************************************************************
1912   * REST_GetUserSelectedTime
1913   *
1914   * Description:
1915   *   This routine display a calendar which allows the user to select
1916   *   a backup time.  If the user selects a time and hits OK, the time
1917   *   selected will be returned,
                              if the user cancels time 0 will be returned.
1918   *
1919   * Parameters:
1920   *   currentWI   (I) - The current work-item to get the time for.
1921   *   currentTime (I) - The current time for the work-item
1922   *   parentWin   (
                  I) - The parent window to make the calendar a child of.
1923   *
1924   * Returns:
1925   *   time_t representing the selected time
1926   *   0 if user cancelled or if an error occurs
1927   *
1928   *****************************************************************/

1930  time_t  REST_GetUserSelectedTime (GREST_Object currentWI,
1931                                    time_t       currentTime,
1932                                    WinPtr       parentWin)

1934  1 {
1935  1   int   selectedHour;                          /* Temporary storage */
1936  1   int   selectedMinutes;                       /* Temporary storage */
1937  1   Char  name[GMAX_OBJECT_LENGTH];              /* Name of the work item */
1938  1   Int   index;                                 /* Loop Counter */
1939  1   Int   month;
                                   /* Month counter to fill cbox with */
1940  1   Int   year;
                                    /* Year counter to fill cbox with */
1941  1   Char  displayString[GMAX_OBJECT_LENGTH]; /* Dispayed year string */
1942  1   int   currentDay;                            /* Today's day */
1943  1   int   currentMonth;                          /* Today's month */
1944  1   int   currentYear;                           /* Today's year */

1946  1   /* Flag that the boxes haven't been initialized yet */
1947  1   boxesInited = BOOL_FALSE;

1949  1   /* Load and initialize the window resources */
1950  1   REST_CalendarLoadInit (parentWin);

1952  1   /* Create the widgets for each day */
1953  1   for (index = 0; index < MAX_DAYS_PER_MONTH; index++)
1954  2   {
1955  2     /* Create a dropdown combo box for the day (
                                                  to show multiple times) */
1956  2     backupTimeBox[index] = REST_CalCreateDateBox ();
1957  2     PANEL_AddWgt (REST_CalendarWindow->CalendarPanel, (
                                                 WgtPtr)backupTimeBox[index]);
1958  2     WGT_SetNfyProc ((WgtPtr)backupTimeBox[index], REST_DateBoxNfy);
1959  2     WGT_SetClientRes ((WgtPtr)backupTimeBox[index], (ClientPtr)index);
1960  2     WGT_SetInvisible ((WgtPtr)backupTimeBox[index]);

1962  2     /* Create a TEd for the day (to show single times) */
1963  2     backupTimeText[index] = REST_CalCreateDateText ();
1964  2     PANEL_AddWgt (REST_CalendarWindow->CalendarPanel, (
                                                 WgtPtr)backupTimeText[index]);
1965  2     WGT_SetClientRes ((WgtPtr)backupTimeText[index], (ClientPtr)index);
1966  2     WGT_SetInvisible ((WgtPtr)backupTimeText[index]);
1967  1   }
```

```
1969  1    /* Get the current date */
1970  1    GTIME_GetCurrentDate (&currentDay, &currentMonth, &currentYear);

1972  1    /* Flag that we have initialized the boxes */
1973  1    boxesInited = BOOL_TRUE;

1975  1    /* If the called passed a time, use it */
1976  1    if (currentTime != 0)
1977  1      currentSelectedTime = currentTime;

1979  1    /* Otherwise, use the current backup time */
1980  1    else if (EDMRST_GetCurrentBackupTime (GREST_Handle,
1981  1                                          &currentSelectedTime) !=
                                                                    E_SUCCESS)
1982  2    {
1983  2      return (0);
1984  1    }

1986  1    /* Break down the time to get the current month and year to display
                                                                          */
1987  1    GTIME_BreakDownTime (currentSelectedTime,
1988  1                         &selectedDay,
1989  1                         &selectedMonth,
1990  1                         &selectedYear,
1991  1                         &selectedHour,
1992  1                         &selectedMinutes);
1993  1    displayedMonth = selectedMonth;
1994  1    displayedYear  = selectedYear;

1996  1    /* Set the window title to the default settings */
1997  1    STR_Cpy (name, EDMRST_GetObjectFullName (GREST_Handle, currentWI));
1998  1    GUTIL_SetDefaultWindowTitle ((WinPtr)REST_CalendarWindow, name);

2000  1    /* Display the current date */
2001  1    REST_UpdateDisplayedDate ();

2003  1    /* Reset the terminated flag */
2004  1    REST_CalTerminated = BOOL_FALSE;

2006  1    /* Position the window and go on our merry way */
2007  1    GWIN_CenterWindowInParent ((WinPtr)REST_CalendarWindow);
2008  1    WIN_ModalProcess ((WinPtr)REST_CalendarWindow);

2010  1    /* Flush the events */
2011  1    EVENT_ProcessPending ();

2013  1    /* Flag that we are done, and terminate the window */
2014  1    REST_CalTerminated = BOOL_TRUE;
2015  1    WIN_Terminate ((WinPtr)REST_CalendarWindow);

2017  1    /* return the selected time */
2018  1    return (currentSelectedTime);
2019    }
```

```
/* -- Template created by NEURON DATA Open Interface.                    */
/* -- Do not alter 'CodeGen' directives.                                 */
/* (( CodeGen: GeneratorVersion 4 ))                                     */

/* -- Code generated on 09/03/99 at 10:48:08.                            */
/* (( CodeGen: CodeHistory ))                                            */


#define ERR_LIB RESTORE

#include <esl/c_portable.h>
#include <esl/ep_xopen.h>
#include <util/esl_core.h>
#include <eerrno/e_errno.h>
#include <util/esl_string.h>

/* WARNING: UNIX DEPENDENCY!!! Used for polling sockets */
#include <stropts.h>
#include <poll.h>
#include <unistd.h>
#include "fbrowser/epcomm_api.h"
#include "util/hyper.h"

#include <restore/restore_api.h>

#include "restDest.h"
#include "restoreP.h"
#include "restUtils.h"
#include "restSelMgr.h"
#include "gutil/winUtils.h"
#include "gutil/guiDefines.h"
#include "gutil/guiUtils.h"
#include "gutil/iconDefs.h"
#include "gutil/iconUtils.h"
#include "gutil/alertMgr.h"



ERR_EXTERN
ERR_INMODULE("restDest")

/* (( CodeGen: ClassImplementationPlaceHolder ))                         */
/* (( CodeGen: WinClassImplementationPlaceHolder ))                      */


/*****************************************************************
 * REST_VerifySpace
 *
 * Description:
 *   This routine will verify the amount of space for the restore.
 *
 * Parameters:
 *   win        (I) - Parent window
 *   hostName   (I) - Name of the destination host
 *   directory  (I) - The destination directory path
 *
 * Returns:
 *   BOOL_TRUE  - If it is OK to proceed with restore
 *   BOOL_FALSE - If there is not enough space and the user cancelled
 *
 *****************************************************************/
```



```
static BoolEnum REST_VerifySpace (RestDestWinPtr win,
                                  Str            hostName,
                                  Str            directory)
{
  struct fs_entry *entries;
                                        /* Mount point entries received */
  int              status;              /* Return status */
  Char             restoreString[64];
                                        /* String for space needed */
  Char             availableString[64];
                                        /* String for space available */
  u_hyper          restoreSize;
                                        /* Space needed for restoral */
  u_hyper          kSize;
                                        /* 1K representation as hyper */
  u_hyper          availableSize;       /* Space available */
  long             restoreFiles;
                                        /* Number of files to be restored */
  long             badFiles;            /* Number of bad files */
  BoolEnum         OKtoRestore = BOOL_TRUE;
                                        /* Flag if OK to proceed */


  /* Validate the parameters */
  if ((hostName != NULL) && (STR_Len (hostName) > 0) &&
      (directory != NULL) && (STR_Len (directory) > 0))
  {

    /* Get the filesystem info for the destination */
    if ((rpc_get_fs_info (hostName,
                          directory,
                          &entries,
                          &status) == SVC_CALL_SUCCEEDED) &&
        (status == E_SUCCESS) &&
        (entries != NULL))
    {
      /* Get the restore size in K */
      EDMRST_GetMarkedTotalSize (GREST_Handle, &restoreSize);
      kSize = ul_to_uh ((unsigned long)1024);
      u_hyper_divide_by (&restoreSize, kSize);

      /* Get the space available (as a hyper) */
      if (entries->kbytes_avail > 0)
        availableSize = ul_to_uh ((
                                   unsigned long) entries->kbytes_avail);
      else
        availableSize = ul_to_uh ((unsigned long) 0);

      /* IF the restore size is greater than the available space */
      if (u_hyper_greater_than (restoreSize, availableSize))
      {
        Char  outputString[MAX_STRING_LENGTH];
                                                /* String to display */

        /* Create the error question */
        REST_SPrintHyper (restoreString, restoreSize);
        REST_SPrintHyper (availableString, availableSize);
        STR_Sprintf (outputString,
                     REST_GetErrorString (REST_SPACE_ERROR_FORMAT),
                     restoreString,
                     availableString);

        /* Ask the user if he/she wants to continue anyways */
        if (GALERT_DisplayQuestion((WinPtr)win,
                                   REST_GetErrorString (
                                                REST_SPACE_ERROR_TITLE),
```

```
                                    GICON_GetWarning(),
                                    outputString,
                                    BOOL_FALSE) != GALERT_Affirmative)

        {
          /* The user cancelled the restore */
          OKtoRestore = BOOL_FALSE;
        }
      }

      /* IF it is still OK to restore */
      if (OKtoRestore)
      {

        /* Get the number of marked files */
        REST_GetCurrentMarkedInfo (&restoreFiles, &badFiles);

        /* IF there is more files then available files (inodes) */
        if (restoreFiles > entries->files_free)
        {
          Char  outputString[MAX_STRING_LENGTH];
                                              /* String to display */

          /* Create the error question */
          STR_Sprintf (restoreString, "%ld", restoreFiles);
          STR_Sprintf (availableString, "%ld", entries->files_free);
          STR_Sprintf (outputString,
                       REST_GetErrorString (REST_INODE_ERROR_FORMAT),
                       restoreString,
                       availableString);

          /* Ask the user if he/she wants to continue anyways */
          if (GALERT_DisplayQuestion((WinPtr)win,
                                    REST_GetErrorString(
                                          REST_INODE_ERROR_TITLE),
                                     GICON_GetWarning(),
                                     outputString,
                                     BOOL_FALSE) !=
                                               GALERT_Affirmative)

          {
            /* The user cancelled the restore */
            OKtoRestore = BOOL_FALSE;
          }
        }
      }

      /* Free the entries returned */
      free_fs_info (&entries);
    }
  }

  /* Return whether or not it is OK to proceed with the restore */
  return (OKtoRestore);
}

/* (( CodeGen: WindowSection Win                                        */
/* ==================================================================== */
/* == Code for Window "Win"                                             */
/* ==================================================================== */

/* (( CodeGen: MenuImplementationPlaceHolder ))
```



```
                                                                     */

/* (( CodeGen: WgtNfyHandler HitAlwaysButtone                        */
static   void C_FAR RestDestWin_HitAlwaysButtone L1(
                                                    RestDestWinPtr, win)
{
}
/* )) CodeGen: WgtNfyHandler HitAlwaysButtone                        */

/* (( CodeGen: WgtNfyHandler HitOlderButton                          */
static   void C_FAR RestDestWin_HitOlderButton L1(RestDestWinPtr, win)
{
}
/* )) CodeGen: WgtNfyHandler HitOlderButton                          */

/* (( CodeGen: WgtNfyHandler HitNeverButton                          */
static   void C_FAR RestDestWin_HitNeverButton L1(RestDestWinPtr, win)
{
}
/* )) CodeGen: WgtNfyHandler HitNeverButton                          */

/* (( CodeGen: WgtNfyHandler HitNetworkButton                        */
static   void C_FAR RestDestWin_HitNetworkButton L1(
                                                    RestDestWinPtr, win)
{
  REST_UpdateDataPath (win);
}
/* )) CodeGen: WgtNfyHandler HitNetworkButton                        */

/* (( CodeGen: WgtNfyHandler HitSymmConnButton                       */
static   void C_FAR RestDestWin_HitSymmConnButton L1(
                                                    RestDestWinPtr, win)
{
  REST_UpdateDataPath (win);
}
/* )) CodeGen: WgtNfyHandler HitSymmConnButton                       */

/* (( CodeGen: WgtNfyHandler HitInPlaceButton                        */
static   void C_FAR RestDestWin_HitInPlaceButton L1(
                                                    RestDestWinPtr, win)
{
  REST_UpdateDestinationSensitivity (win);
}
/* )) CodeGen: WgtNfyHandler HitInPlaceButton                        */

/* (( CodeGen: WgtNfyHandler HitInDirectoryButton                    */
static   void C_FAR RestDestWin_HitInDirectoryButton L1(
                                                    RestDestWinPtr, win)
{
  REST_UpdateDestinationSensitivity (win);
}
/* )) CodeGen: WgtNfyHandler HitInDirectoryButton                    */
```

```
222   /* (( CodeGen: WgtNfyHandler EltSelectedHostBox
                                                                        */
223   static   void C_FAR RestDestWin_EltSelectedHostBox L2(
                        RestDestWinPtr, win, CBoxEltSelectedNfyCPtr, info)
224 1 {
225   }
226   /* )) CodeGen: WgtNfyHandler EltSelectedHostBox
                                                                        */

228   /* (( CodeGen: WgtNfyHandler ValidateDirectoryTEd
                                                                        */
229   static   void C_FAR RestDestWin_ValidateDirectoryTEd L1(
                                                        RestDestWinPtr, win)
230 1 {
231   }
232   /* )) CodeGen: WgtNfyHandler ValidateDirectoryTEd
                                                                        */

234   /* (( CodeGen: WgtNfyHandler HitBrowseButton
                                                                        */
235   static   void C_FAR RestDestWin_HitBrowseButton L1(
                                                        RestDestWinPtr, win)
236 1 {
237 1   REST_GetDestinationDirectory (win);
238   }
239   /* )) CodeGen: WgtNfyHandler HitBrowseButton
                                                                        */

241   /* (( CodeGen: WgtNfyHandler HitOKButton
                                                                        */
242   static   void C_FAR RestDestWin_HitOKButton L1(RestDestWinPtr, win)
243 1 {
244 1   Char              hostName[MAX_CLIENT_NAME_LENGTH];
                                                  /* Host to restore to */
245 1   Char              destHostName[MAX_CLIENT_NAME_LENGTH];
                                                  /* Host to restore to */
246 1   Char              dirName[MAX_STRING_LENGTH];
                                          /* Directory to restore to */
247 1   RestoreInfoPtr    tmpInfo;
                                          /* Pointer to walk parents */

249 1   /* If the user specified the location, update the location */
250 1   if (!TBUT_GetSelected ((TButPtr)win->InPlaceButton))
251 2   {

253 2     /* Get the current destination host */
254 2     STR_Cpy (destHostName, REST_GetCurrentDestHost(win));

256 2     /* Get the directory from the widget */
257 2     STR_Cpy (dirName, TED_GetStr ((TEdPtr)win->DirectoryTEd));

259 2     /* Standardize the pathname to accept NT style paths too e.g.,
                                                              C:\XYZ */
260 2     REST_StandardizePath(dirName);

262 2     /*
263 2      * Get the original host name
264 2      */

266 2     /* Find the client info */
267 2     tmpInfo = currentWorkItemInfo;
268 2     while ((tmpInfo != NULL) && (tmpInfo->type != REST_Client))
269 3     {
270 3       tmpInfo = tmpInfo->parent;
271 2     }
```



```
273 2     /* Make sure we have something */
274 2     if (tmpInfo != NULL)
275 3     {
276 3       STR_Cpy (hostName, tmpInfo->name);
277 2     }
278 2     else
279 3     {
280 3       STR_Cpy (hostName, "");
281 2     }

283 2     /*
284 2      * If a different host or the directory is not blank or "/"
285 2      * verify the space available
286 2      */
287 2     if ((STR_Cmp (destHostName, hostName) != CMP_EQUAL) ||
288 2         ((STR_Cmp (dirName, "") != CMP_EQUAL) &&
289 2          (STR_Cmp (dirName, "/") != CMP_EQUAL)))
290 3     {
291 3       /* Verify that there is enough space to restore to */
292 3       if (!REST_VerifySpace (win, destHostName, dirName))
293 4       {
294 4         return;
295 3       }
296 2     }
297 1   }

299 1   WIN_ModalReturn ((WinPtr)win, (ClientPtr)BOOL_TRUE);
300   }
301   /* )) CodeGen: WgtNfyHandler HitOKButton
                                                                        */

303   /* (( CodeGen: WgtNfyHandler HitCancelButton
                                                                        */
304   static   void C_FAR RestDestWin_HitCancelButton L1(
                                                        RestDestWinPtr, win)
305 1 {
306 1   WIN_ModalReturn ((WinPtr)win, (ClientPtr)BOOL_FALSE);
307   }
308   /* )) CodeGen: WgtNfyHandler HitCancelButton
                                                                        */

310   /* (( CodeGen: WgtNfyHandler HitHelpButton
                                                                        */
311   static   void C_FAR RestDestWin_HitHelpButton L1(RestDestWinPtr, win)
312 1 {
313 1   REST_DestDisplayHelp (win);
314   }
315   /* )) CodeGen: WgtNfyHandler HitHelpButton
                                                                        */
316   /* (( CodeGen: WgtNfyPlaceHolder ))
                                                                        */

318   /* (( CodeGen: UseDefaultNfyHandler name_of_nfy_handler ))
                                                                        */
319   /* (( CodeGen: UseAllDefaultNfyHandlers name_of_wgt_member ))
                                                                        */

321   void      RestDestWin_Construct L1(RestDestWinPtr, win)
322 1 {
323 1         /* ((
              CodeGen: WgtInitializations                               */
324 1         win->PolicyPanel = (PanelPtr)PANEL_GetNamedWgt((
                                                    PanelPtr)win, "PolicyPanel");
325 1         win->AlwaysButtone = (RButPtr)PANEL_GetNamedWgt((
                                                   PanelPtr)win, "AlwaysButtone");
```

```
326 1           win->OlderButton = (RButPtr)PANEL_GetNamedWgt((
                                                PanelPtr)win, "OlderButton");
327 1           win->NeverButton = (RButPtr)PANEL_GetNamedWgt((
                                                PanelPtr)win, "NeverButton");
328 1           win->DataPathPanel = (PanelPtr)PANEL_GetNamedWgt((
                                               PanelPtr)win, "DataPathPanel");
329 1           win->NetworkButton = (RButPtr)PANEL_GetNamedWgt((
                                                PanelPtr)win, "NetworkButton");
330 1           win->SymmConnButton = (RButPtr)PANEL_GetNamedWgt((
                                              PanelPtr)win, "SymmConnButton");
331 1           win->LocationPanel = (PanelPtr)PANEL_GetNamedWgt((
                                                PanelPtr)win, "LocationPanel");
332 1           win->InPlaceButton = (RButPtr)PANEL_GetNamedWgt((
                                                PanelPtr)win, "InPlaceButton");
333 1           win->InDirectoryButton = (RButPtr)PANEL_GetNamedWgt((
                                             PanelPtr)win, "InDirectoryButton");
334 1           win->HostText = (TAreaPtr)PANEL_GetNamedWgt((
                                                  PanelPtr)win, "HostText");
335 1           win->HostBox = (CBoxPtr)PANEL_GetNamedWgt((
                                                   PanelPtr)win, "HostBox");
336 1           win->DirectoryText = (TAreaPtr)PANEL_GetNamedWgt((
                                                PanelPtr)win, "DirectoryText");
337 1           win->DirectoryTEd = (STEdPtr)PANEL_GetNamedWgt((
                                                PanelPtr)win, "DirectoryTEd");
338 1           win->BrowseButton = (PButPtr)PANEL_GetNamedWgt((
                                                PanelPtr)win, "BrowseButton");
339 1           win->ButtonPanel = (PanelPtr)PANEL_GetNamedWgt((
                                                 PanelPtr)win, "ButtonPanel");
340 1           win->OKButton = (PButPtr)PANEL_GetNamedWgt((
                                                    PanelPtr)win, "OKButton");
341 1           win->CancelButton = (PButPtr)PANEL_GetNamedWgt((
                                                PanelPtr)win, "CancelButton");
342 1           win->HelpButton = (PButPtr)PANEL_GetNamedWgt((
                                                  PanelPtr)win, "HelpButton");
343 1           WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->AlwaysButtone, TBUT_NFYHIT,
344 1                   (WinWgtNfyHandlerProc)RestDestWin_HitAlwaysButtone);
345 1           WIN_SetWgtNfyHandler((WinPtr)win, (
                                      WgtPtr)win->OlderButton, TBUT_NFYHIT,
346 1                   (WinWgtNfyHandlerProc)RestDestWin_HitOlderButton);
347 1           WIN_SetWgtNfyHandler((WinPtr)win, (
                                      WgtPtr)win->NeverButton, TBUT_NFYHIT,
348 1                   (WinWgtNfyHandlerProc)RestDestWin_HitNeverButton);
349 1           WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->NetworkButton, TBUT_NFYHIT,
350 1                   (WinWgtNfyHandlerProc)RestDestWin_HitNetworkButton);
351 1           WIN_SetWgtNfyHandler((WinPtr)win, (
                                   WgtPtr)win->SymmConnButton, TBUT_NFYHIT,
352 1                   (WinWgtNfyHandlerProc)RestDestWin_HitSymmConnButton);
353 1           WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->InPlaceButton, TBUT_NFYHIT,
354 1                   (WinWgtNfyHandlerProc)RestDestWin_HitInPlaceButton);
355 1           WIN_SetWgtNfyHandler((WinPtr)win, (
                                WgtPtr)win->InDirectoryButton, TBUT_NFYHIT,
356 1                   (
                      WinWgtNfyHandlerProc)RestDestWin_HitInDirectoryButton);
357 1           WIN_SetWgtNfyHandler((WinPtr)win, (
                                  WgtPtr)win->HostBox, CBOX_NFYELTSELECTED,
358 1                   (WinWgtNfyHandlerProc)RestDestWin_EltSelectedHostBox);
359 1           WIN_SetWgtNfyHandler((WinPtr)win, (
                                 WgtPtr)win->DirectoryTEd, TED_NFYVALIDATE,
360 1                   (
                      WinWgtNfyHandlerProc)RestDestWin_ValidateDirectoryTEd);
361 1           WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->BrowseButton, TBUT_NFYHIT,
362 1                   (WinWgtNfyHandlerProc)RestDestWin_HitBrowseButton);
```



```
363 1           WIN_SetWgtNfyHandler((WinPtr)win, (
                                          WgtPtr)win->OKButton, TBUT_NFYHIT,
364 1                   (WinWgtNfyHandlerProc)RestDestWin_HitOKButton);
365 1           WIN_SetWgtNfyHandler((WinPtr)win, (
                                      WgtPtr)win->CancelButton, TBUT_NFYHIT,
366 1                   (WinWgtNfyHandlerProc)RestDestWin_HitCancelButton);
367 1           WIN_SetWgtNfyHandler((WinPtr)win, (
                                        WgtPtr)win->HelpButton, TBUT_NFYHIT,
368 1                   (WinWgtNfyHandlerProc)RestDestWin_HitHelpButton);
369 1           /* )) CodeGen: WgtInitializations 276537
                                                                          */

371 1     WIN_SetWgtNfyHandler ((WinPtr)win,
372 1                           (WgtPtr)win,
373 1                           WIN_NFYMOUSECLICK,
374 1                           (
                         WinWgtNfyHandlerProc)GUTIL_WinHandleMouseClick);

376       }

378       ClientPtr RestDestWin_LoadProcess L0()
379 1     {
380 1               ClientPtr retval;
381 1               RestDestWinPtr win;

383 1               win = (RestDestWinPtr)WIN_LoadSized("restDest", "Win",
384 1                       sizeof(RestDestWinRec));
385 1               RestDestWin_Construct(win);

387 1               WIN_Init((WinPtr)win);
388 1               retval = WIN_ModalProcess((WinPtr)win);
389 1               WIN_Terminate((WinPtr)win);
390 1               return(retval);
391       }

393       /* )) CodeGen: WindowSection Win
                                                                          */
394       /* (( CodeGen: WindowImplementationPlaceHolder ))
                                                                          */

396       /* (( CodeGen: MainPlaceHolder ))
                                                                          */

398       /*************************************************************************
399        * REST_GetDestinationInfo
400        *
401        * Description:
402        *   This routine will display the destination info dialog to get the
403        *   destination restore options from the user.
404        *
405        * Parameters:
406        *   parentWin     ( I)  - Parent window
407        *   inPlace       ( O)  - Flag for inplace restore
408        *   destHostName (IO)   - The name of the host to restore to
409        *   dirName       (
                          IO)   - The directory on the restore host to restore to
410        *   policy        ( O)  - The overwrite policy
411        *   transport     ( O)  - The restore transport method
412        *
413        * Returns:
414        *   BOOL_TRUE  - If user wants to continue with the restore
415        *   BOOL_FALSE - If user cancelled
416        *
417        *************************************************************************/

419       BoolEnum REST_GetDestinationInfo (WinPtr              parentWin,
```

```
                                        BoolEnum          *inPlace,
                                        Str                destHostName,
                                        Str                dirName,
                                        OverwritePolicy   *policy,
                                        RestoreTransport  *transport)
{
        RestDestWinPtr win;
  BoolEnum       retVal;


        win = (RestDestWinPtr)WIN_LoadSized("restDest", "Win",
                sizeof(RestDestWinRec));
        RestDestWin_Construct(win);

  WIN_Init((WinPtr)win);

  /* Set the Window and Icon Labels */
  GUTIL_AddHostToWindowTitle ((WinPtr)win);

  /* Set the parent window */
  if (parentWin != NULL)
  {
    WIN_SetParentWin ((WinPtr)win, parentWin);
    GWIN_CenterWindowInParent((WinPtr)win);
  }

  /* Set the initial state of the widgets */
  TBUT_SetSelected ((TButPtr)win->InPlaceButton, BOOL_TRUE);
  TBUT_SetSelected ((TButPtr)win->NeverButton, BOOL_TRUE);
  TED_SetStr ((TEdPtr)win->DirectoryTEd, "");

  /* Initialize to SC restore, if not available it will fix itself */
  TBUT_SetSelected ((TButPtr)win->SymmConnButton, BOOL_TRUE);

  REST_UpdateRestoreHosts (win);
  REST_UpdateDestinationSensitivity (win);
  REST_UpdateDataPath (win);

  retVal = (BoolEnum)WIN_ModalProcess((WinPtr)win);

  if (retVal)
  {
    /* Get the current destination host */
    STR_Cpy (destHostName, REST_GetCurrentDestHost(win));

    /* If the user specified the location, update the location */
    *inPlace = TBUT_GetSelected ((TButPtr)win->InPlaceButton);
    if (!*inPlace)
    {
      /* Get the directory from the widget, if allowed  */
      if (TBUT_GetSelected ((TButPtr)win->NetworkButton))
      {
        STR_Cpy (dirName, TED_GetStr ((TEdPtr)win->DirectoryTEd));

        /* Standardize the pathname to accept NT style paths too e.g.,
                                                          C:\XYZ */
        REST_StandardizePath(dirName);

        /* If the directory was left blank,
                                        start in the root directory */
        if (STR_Cmp (dirName, "") == CMP_EQUAL)
          STR_Cpy (dirName, "/");
      }
    }

    /* Get the overwrite policy */
```



```
    if (TBUT_GetSelected ((TButPtr)win->AlwaysButtone))
      *policy = Always_Overwrite;
    else if (TBUT_GetSelected ((TButPtr)win->OlderButton))
      *policy = Older_Only_Overwrite;
    else
      *policy = Never_Overwrite;

    /* Get the transport mechanism */
    if (TBUT_GetSelected ((TButPtr)win->NetworkButton))
      *transport = restoreTransportNetwork;
    else
      *transport = restoreTransportSCSI;
  }

  WIN_Terminate((WinPtr)win);

  EVENT_ProcessPending ();

  return (retVal);
}
```

```
/*****************************************************************
 * restDestMgr.c
 *
 *
 * Copyright 1999 by EMC Corp.
 *
 *
 * Mission Statement:
 *   This file contains the callback functions necessary for the
 *   EDM Restore Destination options window.
 *
 * Required includes:
 *   None
 *
 * Compile-Time Options:
 *   N/A
 *
 *
 * RCS Information:
 *     $RCSfile$
 *     $Revision$
 *     $Date$
 *****************************************************************/

#define  ERR_LIB  RESTORE

/* Muck with the following defines to allow use of putenv which is
                                                          portable */
#define _XOPEN_SOURCE

#include <esl/c_portable.h>
#include <esl/ep_xopen.h>
#include <eerrno/e_errno.h>
#include <util/esl_string.h>

#include <stdlib.h>

#include <libgen.h>
#include <time.h>

#include <tedpub.h>
#include <winpub.h>
#include <strlpub.h>

#include <restore/restore_api.h>
#include "restDest.h"
#include "restore.h"
#include "restoreP.h"
#include "restore/restMgr.h"
#include "restUtils.h"
#include "gutil/iconUtils.h"
#include "gutil/hostUtils.h"
#include "gutil/winUtils.h"
#include "gutil/cboxUtils.h"
#include "gutil/resUtils.h"
#include "gutil/miscUtils.h"
#include "gutil/guiDefines.h"
#include "gutil/guiUtils.h"
#include "gutil/alertMgr.h"
#include "help/helpDefs.h"
#include "help/helpIpc.h"
#include "gutil/fBrowseMgr.h"

ERR_EXTERN
ERR_INMODULE("restore")
```



```
/*************
 * Constants *
 *************/

/*************************
 * Local Global Variables *
 *************************/

/*****************************************************************
 * REST_UpdateDataPath
 *
 * Description:
 *   This routine will update the data path visibility base on what is
 *   allowed for the current work item.
 *
 * Parameters:
 *   None
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void REST_UpdateDataPath (RestDestWinPtr win)
{
  BoolEnum  isSymmOK=TRUE;
  BoolEnum  isNetworkOK=TRUE;
  eerrno_ty status;

  /* Based on the current WI determin if SC restore is OK */
  if ((currentWorkItemInfo != NULL) &&
      (currentWorkItemInfo->restoreObject != NULL))
  {
    /* Make sure the current work item is restorable over SymmConnect
                                                                    */
    if ((status = EDMRST_GetSymmRestoreOption(GREST_Handle,
                                    currentWorkItemInfo->restoreObject,
                                          &isSymmOK)) != E_SUCCESS)
    {
      REST_DisplayErrnoMessage ((WinPtr)win, NULL, NULL, status);

      /* on error, allow SC restore */
      isSymmOK = BOOL_TRUE;
    }

    /* Make sure the current work item is restorable over the network
                                                                    */
    if ((status = EDMRST_GetNetworkRestoreOption(GREST_Handle,
                                    currentWorkItemInfo->restoreObject,
                                          &isNetworkOK)) !=
                                                        E_SUCCESS)
    {
      REST_DisplayErrnoMessage ((WinPtr)win, NULL, NULL, status);

      /* on error, allow network restore */
      isNetworkOK = BOOL_TRUE;
    }
  }
  else
  {
    /*
     * Allow SC and Network to be visible
     * they will be updated more accurately later
```

```
127 2       */

129 2      isSymmOK = BOOL_TRUE;
130 2      isNetworkOK = BOOL_TRUE;
131 1    }

133 1    if (!isNetworkOK)
134 2    {
135 2      /* Not OK to use network, so select the SymmConnect button */
136 2      TBUT_Select ((TButPtr)win->SymmConnButton);
137 1    }

139 1    if (!isSymmOK)
140 2    {
141 2      /* Not OK to use symm connect, so select the network button */
142 2      TBUT_Select ((TButPtr)win->NetworkButton);
143 1    }

145 1    /* Set the visibility of the symm connect button */
146 1    GUTIL_WGT_SetEnabled ((WgtPtr)win->SymmConnButton, isSymmOK);

148 1    /* Set the visibility of the network button */
149 1    GUTIL_WGT_SetEnabled ((WgtPtr)win->NetworkButton, isNetworkOK);

151 1    /* Update the destination sensitivity */
152 1    REST_UpdateDestinationSensitivity (win);
153    }

155    /*****************************************************************
156     * REST_GetCurrentDestHost
157     *
158     * Description:
159     *   This routine will determin the current destination host.
                                                              If the user
160     *   has selected a host it will return that host, otherwise if will
161     *   return the current source host if the user is working on one,
                                                                 otherwise
162     *   it will simply return the local host.
163     *
164     * Parameters:
165     *   None.
166     *
167     * Returns:
168     *   None.
169     *
170     *****************************************************************/

172    Str REST_GetCurrentDestHost (RestDestWinPtr win)
173 1  {
174 1    static Char    destHostName[MAX_CLIENT_NAME_LENGTH];
                                                  /* Current dest host */
175 1    RestoreInfoPtr tmpInfo;
                                                /* Pointer to walk list */

177 1    /* Get the current selected host, if the user has chosen one */
178 1    if (CBOX_HasChoice (win->HostBox))
179 2    {
180 2      STR_Cpy (destHostName, CBOX_ChosenGetLabel(win->HostBox));
181 1    }
182 1    else
183 2    {
184 2      /*
185 2       * Get the current source client
186 2       */

188 2      /* Use the current info */
```



```
189 2      if (currentWorkItemInfo != NULL)
190 2        tmpInfo = currentWorkItemInfo;
191 2      else
192 2        tmpInfo = NULL;

194 2      /* Find the client for this restoral */
195 2      while ((tmpInfo != NULL) && (tmpInfo->type != REST_Client))
196 3      {
197 3        tmpInfo = tmpInfo->parent;
198 2      }

200 2      /* If we found the client get its name */
201 2      if (tmpInfo != NULL)
202 3      {
203 3        STR_Cpy (destHostName, tmpInfo->name);
204 2      }
205 2      else
206 3      {
207 3        /* Didn't find it, use the current host */
208 3        STR_Cpy (destHostName, GUTIL_GetThisHost());
209 2      }
210 1    }

212 1    return (destHostName);
213    }

215    /*****************************************************************
216     * REST_UpdateBrowseButtonSensitivity
217     *
218     * Description:
219     *   This routine sets the sensitivity of the browse button.
220     *
221     *   The browse button is sensitive only if:
222     *     1) The specify location toggle is set
223     *     2) The destination host supports browsing
224     *        (currently Solaris, HP/UX and AIX)
225     *
226     * Parameters:
227     *   None.
228     *
229     * Returns:
230     *   None.
231     *
232     *****************************************************************/

234    void REST_UpdateBrowseButtonSensitivity (RestDestWinPtr win)
235 1  {
236 1    Char     destHostName[MAX_CLIENT_NAME_LENGTH];
                                                  /* Host to restore to */
237 1    BoolEnum          sensitive;
                                          /* Flag if button should be sensitive */
238 1    BuCfgPlatformType platformType;
                                              /* Platform for destination host */
239 1    eerrno_ty         status;

241 1    /* If there is a restore in progress, don't update anything */
242 1    if (REST_RestoreInProgress ())
243 1      return;

245 1    /* Check if specific location is on */
246 1    if (TBUT_GetSelected((TButPtr)win->InDirectoryButton) &&
247 1        TBUT_GetSelected ((TButPtr)win->NetworkButton))
248 2    {
249 2      /* Get the current host */
250 2      STR_Cpy (destHostName, REST_GetCurrentDestHost (win));
```

```
252 2     /* Get the platform type for the host */
253 2     if ((status = EDMRST_GetHostPlatformType (GREST_Handle,
254 2                                               destHostName,
255 2                                               &platformType)) ==
                                                              E_SUCCESS)
256 3     {
257 3       /* Can only browse UNIX or NT clients */
258 3       if ((platformType == BUCFG_PLATFORM_UNIX) ||
259 3           (platformType == BUCFG_PLATFORM_WNT))
260 4       {
261 4         sensitive = BOOL_TRUE;
262 3       }
263 3       /* All others, don't allow browsing */
264 3       else
265 4       {
266 4         sensitive = BOOL_FALSE;
267 3       }
268 2     }
269 2     /* Unable to get platform type */
270 2     else
271 3     {
272 3       REST_DisplayErrnoMessage ((WinPtr)win, NULL, NULL, status);

274 3       /* Assume we can browse,
                                  browser will tell us later if we can't */
275 3       sensitive = BOOL_TRUE;
276 2     }
277 1   }

279 1   /* This is an inplace restore, no browsing necessary */
280 1   else
281 2   {
282 2     sensitive = BOOL_FALSE;
283 1   }

285 1   GUTIL_WGT_SetEnabled ((WgtPtr)win->BrowseButton,  sensitive);
286   }

288   /*********************************************************************
289    * REST_UpdateDestinationSensitivity
290    *
291    * Description:
292    *   This routine set the sensitivity of the user chosen destination
293    *   widgets.  If the user has selected in place, then disable the
294    *   destination options, otherwise enable them.
295    *
296    * Parameters:
297    *   None.
298    *
299    * Returns:
300    *   None.
301    *
302    *********************************************************************/

304   void  REST_UpdateDestinationSensitivity (RestDestWinPtr win)
305 1 {
306 1   BoolEnum isDestinationSelectable;
                                     /* Is the destination selectable */

308 1   /* If there is a restore in progress, don't update anything */
309 1   if (REST_RestoreInProgress ())
310 1     return;

312 1   /* Check if the user in using in place or not */
313 1   isDestinationSelectable = !TBUT_GetSelected((
                                              TButPtr)win->InPlaceButton);
```



```
315 1   /* Enable or disable the widgets appropriately */
316 1   GUTIL_WGT_SetEnabled ((WgtPtr)win->HostText,
317 1                         isDestinationSelectable);
318 1   GUTIL_WGT_SetEnabled ((WgtPtr)win->HostBox,
319 1                         isDestinationSelectable);

321 1   /* If doing a cross restore,
                          enable directory tedd only if not Symm Connect */
322 1   if (TBUT_GetSelected ((TButPtr)win->NetworkButton))
323 2   {
324 2     GUTIL_WGT_SetEnabled ((WgtPtr)win->DirectoryText,
325 2                           isDestinationSelectable);
326 2     GUTIL_WGT_SetEnabled ((WgtPtr)win->DirectoryTEd,
327 2                           isDestinationSelectable);

329 2     /* For Network restores, allow overwrite policy */
330 2     if (!WGT_IsEnabled ((WgtPtr)win->OlderButton))
331 3     {
332 3       TBUT_Select ((TButPtr)win->NeverButton);
333 3       GUTIL_WGT_SetEnabled ((WgtPtr)win->OlderButton, BOOL_TRUE);
334 3       GUTIL_WGT_SetEnabled ((WgtPtr)win->NeverButton, BOOL_TRUE);
335 2     }
336 1   }
337 1   else
338 2   {
339 2     GUTIL_WGT_SetEnabled ((WgtPtr)win->DirectoryText, BOOL_FALSE);
340 2     GUTIL_WGT_SetEnabled ((WgtPtr)win->DirectoryTEd, BOOL_FALSE);

342 2     /* For SC restores, do NOT allow overwrite policy */
343 2     if (WGT_IsEnabled ((WgtPtr)win->OlderButton))
344 3     {
345 3       TBUT_Select ((TButPtr)win->AlwaysButtone);
346 3       GUTIL_WGT_SetEnabled ((WgtPtr)win->OlderButton, BOOL_FALSE);
347 3       GUTIL_WGT_SetEnabled ((WgtPtr)win->NeverButton, BOOL_FALSE);
348 2     }
349 1   }

351 1   /* Check if browsing should be enabled */
352 1   REST_UpdateBrowseButtonSensitivity (win);
353   }

355   /*********************************************************************
356    * REST_UpdateRestoreHosts
357    *
358    * Description:
359    *   This routine will update the restore destination box to the
                                                              current
360    *   choices available.
361    *
362    * Parameters:
363    *   info (I) - Child info record
364    *
365    * Returns:
366    *   None.
367    *
368    *********************************************************************/

370   void  C_FAR  REST_UpdateRestoreHosts (RestDestWinPtr win)
371 1 {
372 1   RestoreInfoPtr  tmpInfo;
                                              /* Pointer to walk parents */
373 1   char           *hosts[HOSTS_BUFFER_LENGTH]; /* Hosts returned */
374 1   long            cookie = INIT_COOKIE;
                                                 /* Ah, the magic cookie */
375 1   short           numEntries;
```

```
                                        /* Number of hosts returned */
  Int             i;                    /* Loop Counter */
  eerrno_ty       eerrno;               /* Error status */

  /* Find the client info */
  tmpInfo = currentWorkItemInfo;
  while ((tmpInfo != NULL) && (tmpInfo->type != REST_Client))
  {
    tmpInfo = tmpInfo->parent;
  }

  /* Make sure we have something */
  if (tmpInfo != NULL)
  {

    /* First, clear out any old hosts: */
    CBOX_GoFirst(win->HostBox);
    while (CBOX_IsOk (win->HostBox))
    {
      CBOX_GoFirst (win->HostBox);
      CBOX_CurRemoveElt (win->HostBox);
    }

    /* Allocate temporary storage for the hosts to be returned */
    for (i = 0; i < HOSTS_BUFFER_LENGTH; i++)
      hosts[i] = (char *) GUTIL_Malloc (
                          MAX_CLIENT_NAME_LENGTH * sizeof(char));

    /* Get all the templates for the new work item */
    while (cookie != DONE_COOKIE)
    {
      numEntries = 0;
      if ((eerrno = EDMRST_GetDestinationHosts (GREST_Handle,
                                                HOSTS_BUFFER_LENGTH,
                                                hosts,
                                                &numEntries,
                                                &cookie)) ==
                                                          E_SUCCESS)
      {

        /* Add all the hosts to the list */
        for (i=0; i<numEntries; i++)
        {
          /* Add this host to the Combo Box, sorted */
          CBOX_AddStrToCbox (win->HostBox, hosts[i]);

          /* By default, select the work-item's source host */
          if (STR_Cmp (tmpInfo->name, hosts[i]) == CMP_EQUAL)
          {
            CBOX_CurSelect (win->HostBox);
          }
        }

      }
      else
      {
        REST_DisplayErrnoMessage ((WinPtr)win, NULL, NULL, eerrno);

        /* Just get out */
        cookie = DONE_COOKIE;
      }
    }

    /* Free up the temporary storage */
    for (i = 0; i < HOSTS_BUFFER_LENGTH; i++)
      GUTIL_Free (hosts[i]);
```



```
  }
}

/*****************************************************************
 * REST_GetDestinationDirectory
 *
 * Description:
 *   This routine will display the filesystem browser window for the
 *   current destination host.  If the user selects a directory from
 *   the browser,
 *                this routine will then update the current destination
 *   directory.
 *
 * Parameters:
 *   None.
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void      REST_GetDestinationDirectory (RestDestWinPtr win)
{
  Char            destHostName[MAX_CLIENT_NAME_LENGTH];
                                                /* Host to restore to */
  Char           *dirName;
                                                /* New Directory */
  Char            currentDirectory[MAX_STRING_LENGTH];
                                                /* Current Directory */

  /* Get the current destination host */
  STR_Cpy (destHostName, REST_GetCurrentDestHost (win));

  /* Get the current directory from the widget */
  STR_Cpy (currentDirectory, TED_GetStr
           ((TEdPtr)win->DirectoryTEd));

  /* Let the user choose a new directory via the browser */
  dirName = GFBROWS_DisplayWin ((WinPtr)win,
                                "Select Restore Destination Directory",
                                destHostName,
                                currentDirectory,
                                NULL,
                                GFBROWS_HIDE_FILES,
                                REST_MODID,
                                REST_DESTINATION_BROWSE);

  /* If the user chose a new directory, set the directory widget */
  if (dirName != NULL)
  {
    TED_SetStr ((TEdPtr)win->DirectoryTEd, dirName);
    GUTIL_Free (dirName);
  }

}

/*****************************************************************
 * REST_DestDisplayHelp
 *
 * Description:
 *   This routine will display help for the destination window
 *
 * Parameters:
 *   None.
 *
 * Returns:
```

```
501      *   None.
502      *
503      ************************************************************************/

505      void REST_DestDisplayHelp (RestDestWinPtr win)
506    1 {
507    1   EDMHELP_Display ((WinPtr)win, REST_MODID, REST_DESTINATION_OPTIONS);
508      }
```

```
/*****************************************************************
 * restFileMgr.c
 *
 *
 * Copyright 1996 by Epoch Systems, Inc.
 *
 *
 * Mission Statement:
 *   This file contains the functions necessary for the file manager
 *   portion of the EDM Restore window.
 *
 * Required includes:
 *   None
 *
 * Compile-Time Options:
 *   N/A
 *
 *
 * RCS Information:
 *     $RCSfile$
 *     $Revision$
 *     $Date$
 *****************************************************************/

#define  ERR_LIB  RESTORE

#include <esl/c_portable.h>
#include <esl/ep_xopen.h>

#include <stdlib.h>

#include <libgen.h>
#include <time.h>

#include <apppub.h>
#include <eventpub.h>
#include <rlibpub.h>
#include <gwpub.h>
#include <respub.h>
#include <lboxpub.h>
#include <mbarpub.h>
#include <panelpub.h>
#include <tareapub.h>
#include <tbutpub.h>
#include <tedpub.h>
#include <winpub.h>
#include <strlpub.h>
#include <drawpub.h>
#include <dsplypub.h>
#include <ibutpub.h>
#include <arraypub.h>

#include "eerrno/e_errno.h"
#include "util/esl_string.h"
#include <restore/restore_api.h>
#include "restore/restMgr.h"
#include "restore.h"
#include "restoreP.h"
#include "restCBMgr.h"
#define REST_FILE_INIT
#include "restFileMgr.h"
#undef REST_FILE_INIT
#include "restCalendar.h"
#include "restSearch.h"
#include "restSelMgr.h"
#include "restUtils.h"
```



```
#include "restAPIUtils.h"
#include "gutil/timeUtils.h"
#include "gutil/iconUtils.h"
#include "gutil/winUtils.h"
#include "gutil/resUtils.h"
#include "gutil/miscUtils.h"
#include "gutil/fileMgr.h"
#include "gutil/tabDefs.h"
#include "gutil/guiDefines.h"
#include "gutil/guiUtils.h"
#include "help/helpDefs.h"
#include "help/helpIpc.h"
#include "gutil/codeTracer.h"
#include "gutil/alertMgr.h"
#include "ipc/ipcl.h"

ERR_EXTERN
ERR_INMODULE("restore")

/*************
 * Constants *
 *************/

#define NUMBER_ITEMS_COLUMNS    9

/************************
 * Local Data Structures *
 ************************/

/*************************
 * Local Global Variables *
 *************************/

/* The file manager context for the restore file manager */
static GFMGR_Context  fileMgrContext;

/* Flags for current state of processing */
static BoolEnum  allowGetChildren = BOOL_TRUE;
static BoolEnum  reReadInProgress = BOOL_FALSE;

/*****************************************************************
 * REST_GetFMgrContext
 *
 * Description:
 *   This routine will return the current file manage context for the
 *   restore window.
 *
 * Parameters:
 *   None.
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

GFMGR_Context REST_GetFMgrContext (void)
{
  return (fileMgrContext);
}
```

```
127  /*****************************************************************
128   * REST_IsReReadInProgress
129   *
130   * Description:
131   *   This routine will determine if a re-read of a work-item is
                                                                  currently
132   *   in progress.
133   *
134   * Parameters:
135   *   None.
136   *
137   * Returns:
138   *   BOOL_TRUE  - If there is a re-read in progress.
139   *   BOOL_FALSE - otherwise
140   *
141   *****************************************************************/
142
143  BoolEnum  REST_IsReReadInProgress (void)
144 1{
145 1  return (reReadInProgress);
146  }
```



```
148  /*****************************************************************
149   * REST_SetReReadInProgress
150   *
151   * Description:
152   *   This routine will set the re-read in progress status to the given
153   *   status.
154   *
155   * Parameters:
156   *   status (I) - Flag if there is a re-read in progress
157   *
158   * Returns:
159   *   None.
160   *
161   *****************************************************************/
162
163  void  REST_SetReReadInProgress (BoolEnum status)
164 1{
165 1  reReadInProgress = status;
166  }
```

```
168   /*****************************************************************
169    * REST_ReReadWorkItem
170    *
171    * Description:
172    *   This routine will re-read all restorable objects for a given
173    *   work item.  It frees all the current children of the work item
174    *   then uses the file manager to re-get the children.  This routine
175    *   attempts to keep the current file manager selection as it was,
176    *   but this is not always possible.
177    *
178    * Parameters:
179    *   currentWI (I) - The work-item info to re-read.
180    *
181    * Returns:
182    *   None.
183    *
184    *****************************************************************/

186   void  REST_ReReadWorkItem (RestoreInfoPtr currentWI)
187 1 {
188 1   RestoreInfoPtr tmpInfo;           /* Temporary info pointer */
189 1   RestoreInfoPtr nextInfo;          /* Pointer to next info */
190 1   RestoreInfoPtr foundInfo = NULL;  /* Info found for selected item */

192 1   /* Free all the current children */
193 1   tmpInfo = currentWI->children;
194 1   while (tmpInfo != NULL)
195 2   {
196 2     nextInfo = tmpInfo->next;
197 2     REST_FreeInfo (tmpInfo);
198 2     tmpInfo = nextInfo;
199 1   }
200 1   currentWI->children = NULL;

202 1   /* Flag that a re-read is in progress */
203 1   REST_SetReReadInProgress (BOOL_TRUE);

205 1   /* Call the file manager to re-get the new children */
206 1   GFMGR_FreezeDisplay (fileMgrContext);

208 1   /* Update the file manager children */
209 1   GFMGR_UpdateChildren (fileMgrContext, currentWI);

211 1   /* Don't allow any reading of children,
                                    only use what we currently have  */
212 1   allowGetChildren = BOOL_FALSE;

214 1   /* If we have any children, try to find the selected child */
215 1   if (currentWI->children != NULL)
216 2   {
217 2     /* Select the object which corresponds to the currently selected
                                                              name */
218 2     foundInfo = REST_FindSelectionString ();
219 1   }
220 1   else
221 2   {
222 2     /* Just use the workitem as the selected item */
223 2     foundInfo = currentWI;
224 1   }


227 1   /* Now select the object */
228 1   if (foundInfo != NULL)
229 2   {
230 2     GFMGR_SelectObject (fileMgrContext, (
```



```
                                                GFMGR_Object)foundInfo, BOOL_FALSE);
231 2     REST_SetSelectionString (REST_GetFullName(foundInfo));
232 1   }
233 1   else
234 2   {
235 2     GFMGR_DeselectAllObjects (fileMgrContext, BOOL_FALSE);
236 1   }

238 1   /* Update the file manager display */
239 1   GFMGR_Display (fileMgrContext);

241 1   /* Flag that the re-read is in done */
242 1   REST_SetReReadInProgress (BOOL_FALSE);
243 1   allowGetChildren = BOOL_TRUE;
244   }
```

```
246  /*****************************************************************
247   * REST_GetBackupCellString
248   *
249   * Description:
250   *   This routine will return the string to display in the given cell
251   *   for the given object.
252   *
253   * Parameters:
254   *   object        (I) - The backup object.
255   *   column        (I) - The column to get the string for.
256   *   returnString  (O) - The string to display.
257   *   justification (O) - The justification to use.
258   *
259   * Returns:
260   *   None.
261   *
262   *****************************************************************/

264  void  REST_GetBackupCellString (GFMGR_Context fMgr,
265                                  GFMGR_Object  object,
266                                  Int           column,
267                                  Str           returnString,
268                                  UInt16        *justification)
269  1 {
270  1   RestoreInfoPtr   info; /* The info record for the object */

272  1   /* Convert the given object to its real type */
273  1   info = (RestoreInfoPtr)object;

275  1   /* Based on the column, get the string */
276  1   switch (column)
277  2   {
278  2     case 4:
279  2       /* If it is a work item draw nothing */
280  2       if (info->type == REST_WorkItem)
281  3       {
282  3         STR_Cpy (returnString, "");
283  2       }
284  2       /* If it is a failed workitem display the error string */
285  2       else if ((info->type == REST_FailedWorkItem) && (
                                          info->errorString != NULL))
286  3       {
287  3         STR_Cpy (returnString, info->errorString);
288  2       }
289  2       /* If it is an error object draw nothing */
290  2       else if (info->type == REST_ErrorObject)
291  3       {
292  3         STR_Cpy (returnString, "");
293  2       }
294  2       /* else draw the access info string */
295  2       else
296  3       {
297  3         STR_Cpy (returnString, REST_GetPermString (info));
298  2       }
299  2       *justification = DRAW_JUSTLEFT | DRAW_JUSTVCENTER;
300  2       break;

302  2      case 5:
303  2       /* If it is a work item draw nothing */
304  2       if (info->type == REST_WorkItem)
305  3       {
306  3         STR_Cpy (returnString, "");
307  2       }
308  2       /* If it is a failed workitem draw nothing */
309  2       else if (info->type == REST_FailedWorkItem)
```



```
310  3       {
311  3         STR_Cpy (returnString, "");
312  2       }
313  2       /* If it is an error object  draw nothing */
314  2       else if (info->type == REST_ErrorObject)
315  3       {
316  3         STR_Cpy (returnString, "");
317  2       }
318  2       /* else draw the owner string */
319  2       else
320  3       {
321  3         STR_Cpy (returnString, REST_GetOwnerString (info));
322  2       }
323  2       *justification = DRAW_JUSTLEFT | DRAW_JUSTVCENTER;
324  2       break;

326  2      case 6:
327  2       /* If it is a work item or a database object draw nothing */
328  2       if (info->type == REST_WorkItem)
329  3       {
330  3         STR_Cpy (returnString, "");
331  2       }
332  2       /* If it is a failed workitem draw nothing */
333  2       else if (info->type == REST_FailedWorkItem)
334  3       {
335  3         STR_Cpy (returnString, "");
336  2       }
337  2       /* If it is an error object  draw nothing */
338  2       else if (info->type == REST_ErrorObject)
339  3       {
340  3         STR_Cpy (returnString, "");
341  2       }
342  2       /* else draw the group string */
343  2       else
344  3       {
345  3         STR_Cpy (returnString, REST_GetGroupString (info));
346  2       }
347  2       *justification = DRAW_JUSTLEFT | DRAW_JUSTVCENTER;
348  2       break;

350  2     case 7:
351  2       /* If it is a work item or a database object draw nothing */
352  2       if (info->type == REST_WorkItem)
353  3       {
354  3         STR_Cpy (returnString, "");
355  2       }
356  2       /* If it is a failed workitem draw nothing */
357  2       else if (info->type == REST_FailedWorkItem)
358  3       {
359  3         STR_Cpy (returnString, "");
360  2       }
361  2       /* If it is an error object  draw nothing */
362  2       else if (info->type == REST_ErrorObject)
363  3       {
364  3         STR_Cpy (returnString, "");
365  2       }
366  2       /* else draw the size string */
367  2       else
368  3       {
369  3         STR_Cpy (returnString, REST_GetSizeString (info));
370  2       }
371  2       *justification = DRAW_JUSTRIGHT | DRAW_JUSTVCENTER;
372  2       break;

374  2     case 8:
375  2       /* If it is a work item or a database object draw nothing */
```

```
376  2        if (info->type == REST_WorkItem)
377  3        {
378  3          STR_Cpy (returnString, "");
379  2        }
380  2        /* If it is a failed workitem draw nothing */
381  2        else if (info->type == REST_FailedWorkItem)
382  3        {
383  3          STR_Cpy (returnString, "");
384  2        }
385  2        /* If it is an error object  draw nothing */
386  2        else if (info->type == REST_ErrorObject)
387  3        {
388  3          STR_Cpy (returnString, "");
389  2        }
390  2        /* else draw the date string */
391  2        else
392  3        {
393  3          STR_Cpy (returnString, REST_GetDateString (info));
394  2        }
395  2        *justification = DRAW_JUSTLEFT | DRAW_JUSTVCENTER;
396  2        break;

398  2      case 9:
399  2        /* If it is a work item or a database object draw nothing */
400  2        if (info->type == REST_WorkItem)
401  3        {
402  3          STR_Cpy (returnString, "");
403  2        }
404  2        /* If it is a failed workitem draw nothing */
405  2        else if (info->type == REST_FailedWorkItem)
406  3        {
407  3          STR_Cpy (returnString, "");
408  2        }
409  2        /* If it is an error object  draw nothing */
410  2        else if (info->type == REST_ErrorObject)
411  3        {
412  3          STR_Cpy (returnString, "");
413  2        }
414  2        /* else draw the time string */
415  2        else
416  3        {
417  3          STR_Cpy (returnString, REST_GetTimeString (info));
418  2        }
419  2        *justification = DRAW_JUSTLEFT | DRAW_JUSTVCENTER;
420  2        break;

422  2      default:
423  2        STR_Cpy (returnString, "");
424  2        *justification = DRAW_JUSTLEFT | DRAW_JUSTVCENTER;
425  2        break;
426  1    }

428    }
```



```
430  /*****************************************************************
431   * REST_ShowPath
432   *
433   * Description:
434   *   This routine will display the full path name for the given object
435   *
436   * Parameters:
437   *   fullPath (I) - The full path to display.
438   *
439   * Returns:
440   *   None.
441   *
442   *****************************************************************/

444  void REST_ShowPath (Str fullPath)
445  1 {
446  1   /* Set the path widget to the given full path */
447  1   TED_SetStr ((TEdPtr)REST_RestoreWin->PathTEd, fullPath);
448    }
```

```
450  /*****************************************************************
451   * REST_ShowObject
452   *
453   * Description:
454   *   This routine will display the given object in the file manager
455   *   and select it.
456   *
457   * Parameters:
458   *   object (I) - The Object to display.
459   *
460   * Returns:
461   *   None.
462   *
463   *****************************************************************/
465  void REST_ShowObject (RestoreInfoPtr object)
466 1{
468 1   /* Validate the object */
469 1  if (object != NULL)
470 2  {
471 2    /* First show the parent object */
472 2    if (object->parent != NULL)
473 3    {
474 3      REST_ShowObject (object->parent);
476 3      /* Open the parent */
477 3      GFMGR_OpenObject (fileMgrContext, object->parent);
478 2    }
480 2    /* Now select this object */
481 2    GFMGR_SelectObject (fileMgrContext, (
                                    GFMGR_Object)object, BOOL_FALSE);
482 2    REST_SetSelectionString (REST_GetFullName(object));
484 2    /* Set the current path to the selected item */
485 2    if ((object->type == REST_Client) || (
                                     object->type == REST_WorkItem))
486 2      REST_ShowPath ("");
487 2    else
488 2      REST_ShowPath (REST_GetFullName(object));
489 1  }
490  }
```



```
492  /*****************************************************************
493   * REST_SetBackupView
494   *
495   * Description:
496   *   This routine will display the object with the given full name
497   *   at the given backup time.
498   *
499   * Parameters:
500   *   backupTime   (I) - The time for the backup to display
501   *   itemFullName (I) - The full name of the object to display
502   *
503   * Returns:
504   *   None.
505   *
506   *****************************************************************/
508  void  REST_SetBackupView (time_t   backupTime,
509                            Str      itemFullName)
510 1{
511 1  RestoreInfoPtr  info;
                                  /* The info for the given name */
512 1  RestoreInfoPtr  selectedObject;    /* Currently selected object */
513 1  eerrno_ty       eerrno;            /* Error status */
514 1  time_t          currentBackupTime; /* Time of the current backup */
515 1  u_long          flags;
517 1  if (TBUT_GetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton))
518 1    flags = BACKUP_SELECTION_FLAG_PARTIAL_OK;
519 1  else
520 1    flags = BACKUP_SELECTION_FLAG_COMPLETE_ONLY;
522 1  /* Get the current backup time */
523 1  if ((eerrno = EDMRST_GetCurrentBackupTime (GREST_Handle,
524 1                                             &currentBackupTime)) !=
                                                                      0)
525 2  {
526 2    /* Hmm... I guess we just use time zero */
527 2    currentBackupTime = 0;
528 1  }
530 1  /* If the backup time is different then the current backup time */
531 1  if ((backupTime != 0) && (backupTime != currentBackupTime))
532 2  {
534 2    /* Set the backup to the given time */
535 2    if ((eerrno = EDMRST_SetBackupForTime(
                                    GREST_Handle, backupTime, flags))== 0)
536 3    {
538 3      /* Update the date */
539 3      if (currentWorkItemInfo != NULL)
540 4      {
541 4        REST_UpdateBackupDate ();
542 3      }
543 2    }
544 2    else
545 3    {
546 3      Char       outputString[2 * GMAX_OBJECT_LENGTH];
                                                  /* String to display */
548 3      /* couldn't switch, create the error message */
549 3      STR_Sprintf (outputString,
550 3                   REST_GetErrorString (
                                        REST_BACKUP_TIME_SWITCH_ERROR),
551 3                   REST_GetTimeDateString (backupTime));
```

```
553  3        /* Display the error message */
554  3        REST_DisplayErrnoMessage ((WinPtr)REST_RestoreWin,
555  3                                   NULL,
556  3                                   outputString,
557  3                                   eerrno);

559  3        /* We're done here */
560  3        return;
561  2      }
562  1    }

564  1    /* Try to find the object in the current work item */
565  1    info = REST_FindInfoInChildren (
                            itemFullName, currentWorkItemInfo, BOOL_TRUE);

567  1    /* If we found it */
568  1    if (info != NULL)
569  2    {
570  2      /* If this is a file object,
                               select its parent to show object in LBOX */
571  2      if ((info->type == REST_File) && (info->parent != NULL))
572  3      {
573  3        selectedObject = info->parent;
574  2      }

576  2      /* Else select this object */
577  2      else
578  3      {
579  3        selectedObject = info;
580  2      }

582  2      /* Show the object in the file manager if not already selected */
583  2      if (!GFMGR_IsObjectSelected (fileMgrContext, (
                                          GFMGR_Object)selectedObject))
584  3      {
585  3        GFMGR_FreezeDisplay (fileMgrContext);
586  3        REST_ShowObject (selectedObject);
587  3        GFMGR_Display (fileMgrContext);

589  3        /* Update the current backup options to the new selected object
                                                                        */
590  3        REST_UpdateBackupOptions (info);
591  2      }

593  2      /* IF the object is not a parent, select it in the list box */
594  2      if (selectedObject != info)
595  2        GFMGR_SelectObjectInLBox (fileMgrContext, (
                                          GFMGR_Object)info, BOOL_TRUE);
596  1    }
597  1    else
598  2    {
599  2      Char       outputString[4 * GMAX_OBJECT_LENGTH];
                                          /* Error string to show */

601  2      /* Create the error string */
602  2      STR_Sprintf (outputString,
603  2                   REST_GetErrorString (REST_SET_VIEW_ERROR),
604  2                   itemFullName);

606  2      /* Display the error message */
607  2      GALERT_DisplayError ((WinPtr)REST_RestoreWin,
608  2                           REST_GetErrorString (REST_ERROR_INDEX),
609  2                           GICON_GetError(),
610  2                           outputString);
611  1    }
```



```
613  }
```

```
615   /*****************************************************************
616    * REST_SetViewToPath
617    *
618    * Description:
619    *   This routine will display the object with the full name currently
620    *   in the path widget in the current backup time.
621    *
622    * Parameters:
623    *   None.
624    *
625    * Returns:
626    *   None.
627    *
628    *****************************************************************/

630   void  REST_SetViewToPath (void)
631 1 {
632 1   Str       itemFullName;
                                    /* The full path to set the view to */
633 1   BoolEnum lastCharFound = BOOL_FALSE;
                                    /* Flag if we verified last char */
634 1   Int       lastChar;
                                    /* Current position of last char */

636 1   /* Get the current path */
637 1   itemFullName = esl_strdup ((Str)TED_GetStr((
                                     TEdPtr)REST_RestoreWin->PathTEd));

639 1   /* Strip off trailing spaces and directory markers */
640 1   REST_StripDirectoryChars (itemFullName);

642 1   /* Set the backup view to this item at the current time */
643 1   REST_SetBackupView (0, itemFullName);

645 1   /* We're done with the name so free it */
646 1   GUTIL_Free (itemFullName);
647   }
```



```
649   /*****************************************************************
650    * REST_GetChildren
651    *
652    * Description:
653    *   This routine will return the children of the parent object via
654    *   the AddChild routine.
655    *
656    * Parameters:
657    *   parent     (I) - The parent to get the children for.
658    *   isSelected (I) - Flag whether or not the object is selected
659    *
660    * Returns:
661    *   None.
662    *
663    *****************************************************************/

665   void REST_GetChildren (GFMGR_Context fMgr,
666                          GFMGR_Object  parent,
667                          BoolEnum      isSelected)
668 1 {
669 1   RestoreInfoPtr  info;
                                    /* The info for the given parent */
670 1   RestoreInfoPtr  tmpObject;
                                  /* Pointer to walk the children with */
671 1   RestoreInfoPtr  originalWI;
                                 /* Save pointer to original work item */
672 1   BoolEnum        found = BOOL_FALSE;
                                    /* Flag whether or not we found it */
673 1   BoolEnum        newWI = BOOL_FALSE;
                                 /* Flag whether or not we have a new WI */
674 1   Int             count = 0;          /* Number of children added */

676 1   /* Convert to the real data type */
677 1   info = (RestoreInfoPtr)parent;

679 1   /* If the info is NULL or it is a file, no children to add */
680 1   if ((info == NULL) || (info->type == REST_File))
681 1     return;

683 1   /* If this object is currently selected,
                                       see if the work-item has changed */
684 1   if ((isSelected) &&
685 1       (!REST_IsReReadInProgress()) &&
686 1       (currentWorkItemInfo != NULL) &&
687 1       (info->type != REST_Client))
688 2   {

690 2     /* Keep a pointer to the original work item */
691 2     originalWI = currentWorkItemInfo;

693 2     /* Find work-item for parent object */
694 2     tmpObject = info;
695 2     while ((!found) && (tmpObject != NULL))
696 3     {
697 3       /* Is this object the work-item */
698 3       if (tmpObject->type == REST_WorkItem)
699 4       {
700 4         /* If this is not the current work-item, make is so */
701 4         if (currentWorkItemInfo != tmpObject)
702 5         {
703 5           currentWorkItemInfo = tmpObject;
704 5           newWI = BOOL_TRUE;
705 4         }
706 4         found = BOOL_TRUE;
707 3       }
```

```
708 3         else
709 4         {
710 4           /* Go to the parent of this object */
711 4           tmpObject = tmpObject->parent;
712 3         }
713 2       }
714 1     }

716 1     /* If this is a different work-item, re-read the work-item data */
717 1     if ((newWI) && (allowGetChildren))
718 2     {
719 2       REST_ReReadWorkItem (currentWorkItemInfo);

721 2       /* Clear the mark flags for all objects */
722 2       REST_ClearObjectMarks (originalWI);
723 1     }
724 1     else
725 2     {

727 2       /*  Create the children if need be, and this is selected */
728 2       if ((info->children == NULL) && (allowGetChildren))
729 3       {
730 3         REST_CreateInfoChildren (info);

732 2       }

734 2       /* Add the children to the file manager */
735 2       tmpObject = info->children;
736 2       while (tmpObject != NULL)
737 3       {

739 3         /*
740 3          * Check if this object should be shown in the file manager
741 3          *   Verify hidden file
742 3          *   Verify offsite file
743 3          *   Verify bad file
744 3          */

746 3         if ((REST_ShowHiddenFiles || (tmpObject->name[0] != '.')) &&
747 3             (REST_ShowBadFiles || !REST_IsBadObject (tmpObject)))
748 4         {
749 4           count++;
750 4           GFMGR_AddChild (fileMgrContext, parent, (
                                                  GFMGR_Object)tmpObject);
751 3         }
752 3         tmpObject = tmpObject->next;
753 2       }
754 1     }

756 1     /* If this is a workitem, update the backup options */
757 1     if ((info->type == REST_WorkItem) &&
758 1         (allowGetChildren) &&
759 1         (isSelected) &&
760 1         (info->children != NULL))
761 2     {
762 2       /* Set the WorkItem options to visible */
763 2       REST_SetFSOptionsVisibility (BOOL_TRUE);
764 2       REST_SetSearchVisiblity (BOOL_TRUE);

766 2       /* Update the partial backup button availability */
767 2       REST_UpdatePartialButton ();

769 2       /* Update the workitem data widgets */
770 2       REST_UpdateBackupTemplates (info);
771 1     }
```



```
773   }
```

```
775  /*********************************************************************
776   * REST_CloseChildren
777   *
778   * Description:
779   *   This routine is provided for the file manager to call when an
780   *   object is closed. It does nothing.
781   *
782   * Parameters:
783   *   parent     (I) - The parent to close.
784   *
785   * Returns:
786   *   None.
787   *
788   *********************************************************************/

790  void REST_CloseChildren (GFMGR_Context fMgr, GFMGR_Object parent)
791 1{
792 1  /* For Speed,
               don't destroy anything until the user closes the window */
793 1  return;
794  }
```



```
796  /*********************************************************************
797   * REST_FindHostInArray
798   *
799   * Description:
800   *   This routine will determine if a given host is in an array of
                                                                      hosts.
801   *
802   * Parameters:
803   *   hostName    (I) - name of the host to look for
804   *   sourceHosts (I) - array of hosts to search
805   *
806   * Returns:
807   *   BOOL_TRUE  - If the host is found in the array
808   *   BOOL_FALSE - otherwise
809   *
810   *********************************************************************/

812  BoolEnum REST_FindHostInArray (Str    hostName,
813                                 char **sourceHosts)
814 1{
815 1  char     **nextHost;           /* Next host to compare */
816 1  BoolEnum   found = BOOL_FALSE; /* Flag if we found it */

818 1  /* Loop through all hosts in the array util we find it */
819 1  nextHost = sourceHosts;
820 1  while ((*nextHost != NULL) && (!found))
821 2  {

823 2    /* See if this one matches */
824 2    if (STR_Cmp (*nextHost, hostName) == CMP_EQUAL)
825 2      found = BOOL_TRUE;
826 2    else
827 2      nextHost++;
828 1  }

830 1  /* Return whether or not we found it */
831 1  return (found);
832  }
```

```
834  /*****************************************************************
835   * REST_AddClientInfo
836   *
837   * Description:
838   *   This routine will add a client info object to the current list.
839   *
840   * Parameters:
841   *   info    (I) - the client info to add.
842   *
843   * Returns:
844   *   None.
845   *
846   *****************************************************************/

848  void REST_AddClientInfo (RestoreInfoPtr *currentList,
849                           RestoreInfoPtr  info)
850 1 {
851 1   RestoreInfoPtr tmpInfo;
                                /* Info to walk the current list with */
852 1   RestoreInfoPtr lastInfo = NULL;    /* Last info we looked at */
853 1   BoolEnum       found = BOOL_FALSE;
                                 /* Flag if we found the insert point */

855 1   /* If this is the first, make it the top */
856 1   if (*currentList == NULL)
857 2   {
858 2     *currentList = info;
859 1   }
860 1   else
861 2   {

863 2     /* Find the insert point */
864 2     tmpInfo = *currentList;
865 2     while ((tmpInfo != NULL) && (!found))
866 3     {

868 3       /* If this is the insert point */
869 3       if (REST_IsLessThan (info, tmpInfo))
870 4       {

872 4         /* Flag that we found the insert point */
873 4         found = BOOL_TRUE;

875 4         /* Insert the new client before the current pointer */
876 4         info->next = tmpInfo;
877 4         if (lastInfo != NULL)
878 4           lastInfo->next = info;
879 4         else
880 4           *currentList = info;
881 3       }
882 3       else
883 4       {
884 4         /* Not this one, save the pointer and go to the next */
885 4         lastInfo = tmpInfo;
886 4         tmpInfo = tmpInfo->next;
887 3       }
888 2     }

890 2     /* If we didn't find an insert point, tack it onto the end */
891 2     if (!found)
892 3     {
893 3       lastInfo->next = info;
894 2     }
895 1   }
896  }
```



```
898  /*****************************************************************
899   * REST_ShowClients
900   *
901   * Description:
902   *   This routine will add clients to the file manager and to the
903   *   current object list.  It will use any clients passed on the
904   *   command line that are indeed backup clients.  If no clients
905   *   are passed on the command line, it will display all restorable
906   *   clients. The list given will be updated to the list of valid
907   *   clients being shown.
908   *
909   *   This routine will then select the workites that are in the
910   *   given work item list.
911   *
912   *   NOTE:
913   *         Currently only one work item may be selected at a time.
914   *         This limitation causes this routine to only select the
915   *         first work item in the given list.
916   *
917   * Parameters:
918   *   clientList (
                 IO) - List of clients to show, updated to valid clients
919   *   wiList     (I)  - List of work items to select
920   *
921   * Returns:
922   *   None.
923   *
924   *****************************************************************/

926  void  REST_ShowClients (RestoreClientPtr   *clientList,
927                          RestoreWorkItemPtr  wiList)
928 1 {
929 1   RestoreInfoPtr   topInfo = NULL;
                                          /* Beginning of client list */
930 1   RestoreClientPtr thisClient;          /* Client info in the list */
931 1   RestoreClientPtr nextClient;
                                        /* Next client info in the list */
932 1   RestoreClientPtr lastClient;
                                        /* Last client info in the list */
933 1   Char             clientName[MAX_CLIENT_NAME_LENGTH];
                                             /* Next client to add */
934 1   RestoreInfoPtr   info;
                                      /* New info object for the client */
935 1   RestoreInfoPtr   tmpInfo;
                                       /* Pointer to walk the list with */
936 1   RestoreInfoPtr   nextInfo;
                                       /* Pointer to next item in list */
937 1   RestoreInfoPtr   lastInfo;
                                       /* Pointer to last item in list */
938 1   long             cookie = INIT_COOKIE; /* Ah, the magic cookie */
939 1   short            numEntries;
                                          /* Number of hosts returned */
940 1   char            *tempHosts[HOSTS_BUFFER_LENGTH];
                                             /* Temporary host array */
941 1   char           **sourceHosts;          /* Array of hosts */
942 1   short            hostCount = 0;
                                        /* Count of restorable hosts */
943 1   Int              index;               /* Loop Counter */
944 1   eerrno_ty        eerrno;              /* Error status */
945 1   BoolEnum         invalidClient = BOOL_FALSE;
                                        /* Any clients found invalid */
946 1   static BoolEnum  isValidated = BOOL_FALSE;
                                      /* Have we validated the clients */
947 1   BoolEnum         wiFound = BOOL_FALSE;
                                       /* Flag if we found/selected a WI */
```

```
948 1   RestoreWorkItemPtr nextWi;
                                    /* Next WI to find from given list */
949 1   RestoreInfoPtr   wiInfo;
                                    /* The work item that was found */
950 1   RestoreInfoPtr   clientInfo;
                                    /* Next client info in the list */

952 1   if (!isValidated)
953 2   {

955 2     isValidated = BOOL_TRUE;

957 2     /*
958 2      * Get the number of restorable hosts
959 2      */

961 2     /* Allocate space for the temporary hosts */
962 2     for (index=0; index < HOSTS_BUFFER_LENGTH; index++)
963 2       tempHosts[index] = (char *) GUTIL_Malloc (
                                  MAX_CLIENT_NAME_LENGTH * sizeof(char));

965 2     /* Get the source hosts and count them */
966 2     while (cookie != DONE_COOKIE)
967 3     {
968 3       if ((eerrno = EDMRST_GetSourceHosts (GREST_Handle,
969 3                                            REST_RestoreClient,
970 3                                            HOSTS_BUFFER_LENGTH,
971 3                                            tempHosts,
972 3                                            &numEntries,
973 3                                            &cookie)) == 0)
974 4       {
975 4         /* Add this count to the host count */
976 4         hostCount += numEntries;
977 3       }
978 3       else
979 4       {
980 4         /* Got an error, ignore it and get out */
981 4         cookie = DONE_COOKIE;
982 3       }
983 2     }

985 2     /* We're done with the temporary hosts, free 'em up */
986 2     for (index=0; index < HOSTS_BUFFER_LENGTH; index++)
987 2       GUTIL_Free (tempHosts[index]);

989 2     /* Now, really get the restorable hosts */
990 2     if (hostCount > 0)
991 3     {

993 3       /* Allocate enough space for the restorable hosts */
994 3       sourceHosts = (char **) GUTIL_Malloc ((hostCount + 1) * sizeof(
                                                              char *));
995 3       for (index=0; index < hostCount; index++)
996 3         sourceHosts[index] = (char *) GUTIL_Malloc (
                                  MAX_CLIENT_NAME_LENGTH * sizeof(char));

998 3       /* Set the last one to NULL */
999 3       sourceHosts[hostCount] = NULL;

1001 3      /* Get the hosts (
                          no need to loop, we know how many there are) */
1002 3      cookie = INIT_COOKIE;
1003 3      EDMRST_GetSourceHosts (GREST_Handle,
1004 3                             REST_RestoreClient,
1005 3                             hostCount,
1006 3                             sourceHosts,
```



```
1007 3                             &numEntries,
1008 3                             &cookie);

1010 3      /* Set the last client */
1011 3      lastClient = *clientList;

1013 3      /* If no clients are currently selected, show all */
1014 3      if (*clientList != NULL)
1015 4      {
1016 4        /* Loop through the current hosts,
                                          and add only the backup clients */
1017 4        thisClient = *clientList;
1018 4        while (thisClient != NULL)
1019 5        {

1021 5          /* Save a pointer to the next client */
1022 5          nextClient = thisClient->next;

1024 5          /* Check if this is a backup client */
1025 5  /*
1026 5          if (REST_FindHostInArray (
                                  thisClient->clientName, sourceHosts) &&
1027 5              REST_ValidateClient (thisClient->clientName))
1028 5  */
1029 5          if (REST_FindHostInArray (
                                   thisClient->clientName, sourceHosts))
1030 6          {
1031 6            /* Create the new client info and add it to the list */
1032 6            info = REST_CreateClientInfo(thisClient->clientName);
1033 6            REST_AddClientInfo (&topInfo, info);
1034 6            lastClient = thisClient;
1035 5          }
1036 5          else
1037 6          {
1038 6            if (lastClient != NULL)
1039 6              lastClient->next = thisClient->next;
1040 6            else
1041 6              *clientList = thisClient->next;

1043 6            /* Free this client */
1044 6            GUTIL_Free (thisClient);

1046 6            /* Flag if the client was invalid */
1047 6            if (REST_FindHostInArray (
                                   thisClient->clientName, sourceHosts))
1048 7            {
1049 7              invalidClient = BOOL_TRUE;
1050 6            }
1051 5          }

1053 5          /* Go to the next host */
1054 5          thisClient = nextClient;
1055 4        }
1056 3      }

1058 3      /*
1059 3       * If no clients were selected or none selected are restoreable
1060 3       *   Just show all available hosts (better than none)
1061 3       */

1063 3      if (topInfo == NULL)
1064 4      {

1066 4        /* Loop through all the source hosts */
1067 4        for (index = 0; index < hostCount; index++)
1068 5        {
```

```
1069  5  /*
1070  5            if (REST_ValidateClient (sourceHosts[index]))
1071  5  */
1072  5  if (BOOL_TRUE)
1073  6          {
1074  6            /* Create the new client info and add it to the list */
1075  6            info = REST_CreateClientInfo(sourceHosts[index]);
1076  6            REST_AddClientInfo (&topInfo, info);

1078  6            /* Create the new client */
1079  6            nextClient = (RestoreClientPtr) GUTIL_Malloc (sizeof(
                                                        RestoreClientRec));
1080  6            STR_Cpy (nextClient->clientName, sourceHosts[index]);
1081  6            nextClient->next = NULL;

1083  6            /* Update the client list */
1084  6            if (lastClient != NULL)
1085  6              lastClient->next = nextClient;
1086  6            else
1087  6              *clientList = nextClient;

1089  6            lastClient = nextClient;
1090  5          }
1091  5          else
1092  6          {
1093  6            invalidClient = BOOL_TRUE;
1094  5          }
1095  4        }
1096  3      }

1098  3      /* Free up the hosts and the data */
1099  3      for (index=0; index < hostCount; index++)
1100  3        GUTIL_Free (sourceHosts[index]);
1101  3      GUTIL_Free (sourceHosts);
1102  2    }
1103  1  }
1104  1  else
1105  2  {
1106  2    /* Loop through the current hosts we know they are all valid */
1107  2    thisClient = *clientList;
1108  2    while (thisClient != NULL)
1109  3    {

1111  3      /* Create the new client info and add it to the list */
1112  3      info = REST_CreateClientInfo(thisClient->clientName);
1113  3      REST_AddClientInfo (&topInfo, info);

1115  3      /* Go to the next host */
1116  3      thisClient = thisClient->next;
1117  2    }
1118  1  }

1120  1  /* If there are still no clients restoreable */
1121  1  if (topInfo == NULL)
1122  2  {
1123  2    /* If there were only invalid clients,
                                          display the no data error */
1124  2    if (invalidClient)
1125  3    {
1126  3      GALERT_DisplayError ((WinPtr)REST_RestoreWin,
1127  3                           REST_GetErrorString (REST_ERROR_INDEX),
1128  3                           GICON_GetError(),
1129  3                           REST_GetErrorString (
                                        REST_NO_RESTORE_DATA_ERROR));
1130  2    }
1131  2    /* Else, display the permission denied error */
```



```
1132  2    else
1133  3    {
1134  3      GALERT_DisplayError ((WinPtr)REST_RestoreWin,
1135  3                           REST_GetErrorString (REST_ERROR_INDEX),
1136  3                           GICON_GetError(),
1137  3                           REST_GetErrorString (
                                          REST_PERMISSION_ERROR));
1138  2    }
1139  2    GUTIL_Shutdown ();
1140  1  }

1142  1  /* Add all the clients to the File Manager */
1143  1  tmpInfo = topInfo;
1144  1  while (tmpInfo != NULL)
1145  2  {
1146  2    GFMGR_AddChild (fileMgrContext, NULL, (GFMGR_Object)tmpInfo);
1147  2    tmpInfo = tmpInfo->next;
1148  1  }

1150  1  /* If any workitems were passed in, find and select them */
1151  1  if (wiList != NULL)
1152  2  {
1153  2    /*
1154  2     * NOTE:
1155  2     *
1156  2     * This should loop through all selected work items when we are
1157  2     * able to multi-select work items.
1158  2     */

1160  2    nextWi = wiList;
1161  2    clientInfo = topInfo;
1162  2    while (!wiFound && (clientInfo != NULL))
1163  3    {
1164  3      REST_CreateInfoChildren (clientInfo);
1165  3      wiInfo = REST_FindInfoInChildren (
                                  nextWi->wiName, clientInfo, BOOL_TRUE);
1166  3      if (wiInfo != NULL)
1167  4      {
1168  4        /* Flag that we found the work item */
1169  4        wiFound = BOOL_TRUE;

1171  4        /* Show the workitems for this client in the file manager */
1172  4        GFMGR_OpenObject (fileMgrContext, (GFMGR_Object)clientInfo);
1173  3      }
1174  3      else
1175  4      {
1176  4        /* Go on to the next client */
1177  4        clientInfo = clientInfo->next;
1178  3      }
1179  2    }
1180  1  }

1182  1  /* Select the first host, unless a work item was selected */
1183  1  if (!wiFound)
1184  2  {
1185  2    GFMGR_SelectObject (fileMgrContext, topInfo, BOOL_TRUE);
1186  2    REST_UpdateBackupOptions (NULL);
1187  1  }
1188  1  else
1189  2  {
1190  2    /* Select the work item in the file manager */
1191  2    GFMGR_SelectObject (fileMgrContext, (
                                        GFMGR_Object)wiInfo, BOOL_TRUE);
1192  2    REST_UpdateBackupOptions (wiInfo);
1193  1  }
1194  }
```

```
1196  /*****************************************************************
1197   * REST_VerifySelection
1198   *
1199   * Description:
1200   *   This routine is provided for the file manager.  It determines
1201   *   if it is OK to select the given object.  We must verify with
1202   *   the user before switching WIs if anything is marked or
1203   *   if the search window is displayed (
                                          due to the restore API limitation
1204   *   of one workitem at a time).
1205   *
1206   * Parameters:
1207   *   selectedObject (I) - Object to verify selection for
1208   *   isMultiSelect  (I) - Flag is this is a multi select
1209   *
1210   * Returns:
1211   *   BOOL_TRUE  - If it is OK to select the object
1212   *   BOOL_FALSE - otherwise
1213   *
1214   *****************************************************************/

1216  BoolEnum REST_VerifySelection (GFMGR_Context fMgr,
1217                                 GFMGR_Object  selectedObject,
1218                                 BoolEnum      isMultiSelect,
1219                                 BoolEnum      isOpenObject)
1220  1 {
1221  1   RestoreInfoPtr   info;
                                 /* Real representation of the object */
1222  1   RestoreInfoPtr   tmpObject;
                                    /* Pointer to walk the list with */
1223  1   RestoreInfoPtr   newWorkItem = NULL;
                                 /* Work item for the selected object */
1224  1   BoolEnum         found = BOOL_FALSE; /* Flag if we found it */
1225  1   Char             outputString[MAX_STRING_LENGTH];
                                                      /* Error string */
1226  1   BoolEnum         removeSearch = BOOL_FALSE;
                                       /* Should search be removed */
1227  1   BoolEnum         returnStatus = BOOL_TRUE;  /* Status to return */

1229  1   /* If a restore is in progress, no selecting is allowed! */
1230  1   if (REST_RestoreInProgress() || REST_SearchInProgress())
1231  1     return (BOOL_FALSE);

1233  1   /* Get the object in the real data type */
1234  1   info = (RestoreInfoPtr)selectedObject;

1236  1   /* Go to the top of the selected box,
                                    so we can check if anything is marked */
1237  1   LBOX_GoHome (REST_RestoreWin->SelectedListBox);

1239  1   /*
1240  1    * Quick check,
1241  1    *  If the user selected nothing it is always OK to switch.
1242  1    *  If the search window is not displayed and nothing is marked it
                                                                      is
1243  1    *  also OK to switch.
1244  1    */

1246  1   if ((info == NULL) ||
1247  1       (!REST_SearchWindowIsDisplayed () &&
1248  1        LBOX_CurGetClientData(
                              REST_RestoreWin->SelectedListBox) == NULL))
1249  2   {
1250  2     return (BOOL_TRUE);
1251  1   }
```



```
1253  1   /*
1254  1    * Quick check,
1255  1    *  If the user selected a client and it is the current client
1256  1    *  it is OK to switch.
1257  1    */

1259  1   if ((info->type == REST_Client) && (
                                        info == REST_GetCurrentClientInfo ()))
1260  2   {
1261  2     return (BOOL_TRUE);
1262  1   }

1264  1   /* Find the work-item object for this object */
1265  1   tmpObject = info;
1266  1   while ((newWorkItem == NULL) && (tmpObject != NULL))
1267  2   {
1268  2     /* If this is the work-item object we're done */
1269  2     if (tmpObject->type == REST_WorkItem)
1270  3     {
1271  3       newWorkItem = tmpObject;
1272  2     }

1274  2     /* Else try its parent */
1275  2     else
1276  3     {
1277  3       tmpObject = tmpObject->parent;
1278  2     }
1279  1   }

1281  1   /* If the search window is displayed, ask user first */
1282  1   if (REST_SearchWindowIsDisplayed () &&
1283  1       (newWorkItem != currentWorkItemInfo))
1284  2   {
1285  2     if (info->type == REST_Client)
1286  3     {
1287  3       STR_Cpy (outputString,
1288  3                REST_GetErrorString (
                                      REST_SWITCH_CLIENT_SEARCH_WARNING));
1289  2     }
1290  2     else
1291  3     {
1292  3       STR_Cpy (outputString,
1293  3                REST_GetErrorString (REST_SWITCH_WIS_SEARCH_WARNING));
1294  2     }

1296  2     if (GALERT_DisplayQuestion ((WinPtr)REST_RestoreWin,
1297  2                                 REST_GetErrorString (
                                                  REST_WARNING_INDEX),
1298  2                                 GICON_GetWarning(),
1299  2                                 outputString,
1300  2                                 BOOL_FALSE) == GALERT_Affirmative)
1301  3     {
1302  3       removeSearch = BOOL_TRUE;
1303  2     }
1304  2     else
1305  3     {
1306  3       return (BOOL_FALSE);
1307  2     }
1308  1   }

1310  1   /*
1311  1    * If something is marked, verify if we have switched WIs
1312  1    */

1314  1   if (LBOX_CurGetClientData(REST_RestoreWin->SelectedListBox) != NULL)
```

```
1315 2    {
1316
1317 2      /* For client selection, ask user before switching clients */
1318 2      if (info->type == REST_Client)
1319 3      {
1320
1321 3        if (GALERT_DisplayQuestion ((WinPtr)REST_RestoreWin,
1322 3                                    REST_GetErrorString (
                                                REST_WARNING_INDEX),
1323 3                                    GICON_GetWarning(),
1324 3                                    REST_GetErrorString (
                                                REST_SWITCH_CLIENT_WARNING),
1325 3                                    BOOL_FALSE) != GALERT_Affirmative)
1326 4        {
1327 4          returnStatus = BOOL_FALSE;
1328 3        }
1329 2      }
1330
1331 2      /* If we are looking at a work-item already,
                                          see if it is the same one */
1332 2      else if (currentWorkItemInfo != NULL)
1333 3      {
1334
1335 3        /* If this is a different Work-Item, ask before switching */
1336 3        if (newWorkItem != currentWorkItemInfo)
1337 4        {
1338 4          if (GALERT_DisplayQuestion ((WinPtr)REST_RestoreWin,
1339 4                                      REST_GetErrorString (
                                                  REST_WARNING_INDEX),
1340 4                                      GICON_GetWarning(),
1341 4                                      REST_GetErrorString (
                                                    REST_SWITCH_WIS_WARNING),
1342 4                                      BOOL_FALSE) != GALERT_Affirmative)
1343 5          {
1344 5            returnStatus = BOOL_FALSE;
1345 4          }
1346 3        }
1347 2      }
1348 1    }
1349
1350 1    /* Remove the search dialog if necessary */
1351 1    if (returnStatus && removeSearch)
1352 1      REST_SearchRemove ();
1353
1354 1    /* Return the determined status */
1355 1    return (returnStatus);
1356   }
```



```
1358  /*****************************************************************
1359   * REST_LBoxSelectionNfy
1360   *
1361   * Description:
1362   *   This routine will sensitize and desensize the mark and unmark
                                                                  buttons
1363   *   based on the file manager list box selection status.
1364   *
1365   * Parameters:
1366   *   None.
1367   *
1368   * Returns:
1369   *   None.
1370   *
1371   *****************************************************************/
1372
1373  void REST_LBoxSelectionNfy (GFMGR_Context fMgr)
1374  1 {
1375  1   RestoreInfoPtr   info;
                                    /* Next selected info in the lbox */
1376  1   BoolEnum         mark = BOOL_FALSE;
                                     /* Senitivity of the mark button */
1377  1   BoolEnum         unmark = BOOL_FALSE;
                                   /* Senitivity of the unmark button */
1378  1   boolean_ty       isMarkable = FALSE;
                                  /* Flag if object is markable */
1379  1   Int              numWorkItems = 0;
                                    /* Number of workitems selected */
1380
1381  1   /* If a restore is in progress, don't update any sensitivity */
1382  1   if (REST_RestoreInProgress() || REST_SearchInProgress())
1383  1     return;
1384
1385  1   /* Get all items that are highlighted */
1386  1   info = (RestoreInfoPtr)GFMGR_GetFirstSelectedLBoxObject (
                                                          fileMgrContext);
1387  1   while (info != NULL)
1388  2   {
1389
1390  2     /* If this object is marked, sensitize the unmark button */
1391  2     if (info->marked)
1392  3     {
1393  3       unmark = BOOL_TRUE;
1394  2     }
1395
1396  2     /* Else If this is a restore object, check if it is markable */
1397  2     else if (info->restoreObject != NULL)
1398  3     {
1399  3       if (info->type == REST_WorkItem)
1400  4       {
1401  4         mark = BOOL_TRUE;
1402  4         numWorkItems++;
1403  3       }
1404  3       else if (GREST_IsObjectMarkable (
                                    GREST_Handle, info->restoreObject) &&
1405  3                ((info->status != Backup_Bad) || REST_MarkBadFiles))
1406  4       {
1407  4         /* If this object is markable, sensitize the mark button */
1408  4         mark = BOOL_TRUE;
1409  3       }
1410  2     }
1411
1412  2     /* get the next selected row */
1413  2     info = (RestoreInfoPtr)GFMGR_GetNextSelectedLBoxObject (
                                                          fileMgrContext);
```

```
1414 1    }

1416 1    /* Set the sensitivity of the buttons */
1417 1    mark &= (numWorkItems <= 1);
1418 1    WGT_SetEnabled ((WgtPtr)REST_RestoreWin->MarkButton, mark);
1419 1    WGT_SetEnabled ((WgtPtr)REST_RestoreWin->UnmarkButton, unmark);
1420    }
```



```
1422  /*****************************************************************
1423   * REST_FileSelectionNfy
1424   *
1425   * Description:
1426   *   This routine will handle the selection of an object in the file
1427   *   manager scrolled area.
1428   *
1429   * Parameters:
1430   *   selectedObject (I) - the object that was selected.
1431   *
1432   * Returns:
1433   *   None.
1434   *
1435   *****************************************************************/

1437  void REST_FileSelectionNfy (GFMGR_Context fMgr,
1438                              BoolEnum      isOpenObject)
1439 1 {
1440 1   RestoreInfoPtr    info;
                                /* The real data type for the object */
1441 1   eerrno_ty         eerrno;     /* Error code */
1442 1   RestoreObjectType objectType; /* Object type of the object */
1443 1   Str               fullPath;
                               /* Copy of the full path of the object */

1445 1   /* If we are sorting or re-reading, do nothing */
1446 1   if (REST_IsReReadInProgress())
1447 1     return;

1449 1   /* Get the real info */
1450 1   info = (RestoreInfoPtr) GFMGR_GetFirstSelectedObject (
                                                          fileMgrContext);

1452 1   /* Verify that somethin is selected */
1453 1   if (info != NULL)
1454 2   {

1456 2     /*
1457 2      * Grab a copy of the full path and type incase we change
                                                          workitems and
1458 2      * invalidate the current objects
1459 2      */

1461 2     fullPath = esl_strdup (REST_GetFullName(info));
1462 2     objectType = info->type;

1464 2     /* Save the name of the object */
1465 2     REST_SetSelectionString (fullPath);

1467 2     /* Update the backup options based on this object */
1468 2     REST_UpdateBackupOptions (info);

1470 2     /* Update buttons based on what is selected in the LBox */
1471 2     REST_LBoxSelectionNfy (NULL);

1473 2     /* Set the current path to the selected item */
1474 2     if ((objectType == REST_Client) || (objectType == REST_WorkItem))
1475 2       REST_ShowPath ("");
1476 2     else
1477 2       REST_ShowPath (fullPath);

1479 2     GUTIL_Free (fullPath);
1480 1   }
1481  }
```

```
1483  /*****************************************************************
1484   * REST_IsMarkable
1485   *
1486   * Description:
1487   *   This routine will determine if the given object is 'markable'.
1488   *
1489   * Parameters:
1490   *   fileObject (I) - the object to check out
1491   *
1492   * Returns:
1493   *   None.
1494   *
1495   *****************************************************************/

1497  static BoolEnum REST_IsMarkable( GFMGR_Context fMgr,
1498                                   GFMGR_Object  fileObject )
1499 1 {
1500 1   RestoreInfoPtr info;      /* The real info for the object */
1501 1   BoolEnum       retVal;    /* Value to return */

1503 1   /* Cast back to the real object */
1504 1   info = (RestoreInfoPtr)fileObject;

1506 1   /* Clients workitems and errors are not markable */
1507 1   if ((info->type == REST_Client) ||
1508 1       (info->type == REST_FailedWorkItem) ||
1509 1       (info->type == REST_ErrorObject))
1510 2   {
1511 2     retVal = BOOL_FALSE;
1512 1   }
1513 1   else
1514 2   {
1515 2     retVal = BOOL_TRUE;
1516 1   }

1518 1   return (retVal);
1519   }
```



```
1521  /*****************************************************************
1522   * REST_IsMarked
1523   *
1524   * Description:
1525   *   This routine will determine if the given object is 'marked'.
1526   *
1527   * Parameters:
1528   *   fileObject (I) - the object to check out
1529   *
1530   * Returns:
1531   *   None.
1532   *
1533   *****************************************************************/

1535  static BoolEnum REST_IsMarked( GFMGR_Context fMgr,
1536                                 GFMGR_Object  fileObject )
1537 1 {
1538 1   RestoreInfoPtr info;      /* The real info for the object */

1540 1   /* Cast back to the real object */
1541 1   info = (RestoreInfoPtr)fileObject;

1543 1   return (info->marked);
1544   }
```

```
1546  /*************************************************************************
1547   * REST_HasMarkedChildren
1548   *
1549   * Description:
1550   *   This routine will determine if the given object has any marked
                                                                    children.
1551   *   This only returns true if the children exist,
                                                it does not retrieve new
1552   *   children.
1553   *
1554   * Parameters:
1555   *   parent (I) - the parent to check
1556   *
1557   * Returns:
1558   *   None.
1559   *
1560   *************************************************************************/

1562  static BoolEnum REST_HasMarkedChildren (RestoreInfoPtr parent)
1563 1{
1564 1  BoolEnum       retVal = BOOL_FALSE;
1565 1  RestoreInfoPtr nextChild;

1567 1  nextChild = parent->children;
1568 1  while (!retVal && (nextChild != NULL))
1569 2  {
1570 2    if (nextChild->marked)
1571 3    {
1572 3      retVal = BOOL_TRUE;
1573 2    }
1574 2    else
1575 3    {
1576 3      retVal = REST_HasMarkedChildren (nextChild);
1577 3      nextChild = nextChild->next;
1578 2    }
1579 1  }

1581 1  return (retVal);
1582  }
```



```
1584  /*************************************************************************
1585   * REST_IsPartialMarked
1586   *
1587   * Description:
1588   *   This routine will determine if the given object is 'marked'.
1589   *
1590   * Parameters:
1591   *   fileObject (I) - the object to check out
1592   *
1593   * Returns:
1594   *   None.
1595   *
1596   *************************************************************************/

1598  static BoolEnum REST_IsPartialMarked( GFMGR_Context fMgr,
1599                                        GFMGR_Object  fileObject )
1600 1{
1601 1  RestoreInfoPtr info;      /* The real info for the object */
1602 1  BoolEnum       retVal = BOOL_FALSE;
1603 1  RestoreInfoPtr nextInfo;
1604 1  RestoreInfoPtr childInfo;
1605 1  Str            fullName;

1607 1  /* Cast back to the real object */
1608 1  info = (RestoreInfoPtr)fileObject;

1610 1  if (info->restoreObject != NULL)
1611 2  {
1612 2    /* Determine if any existing children are marked */
1613 2    retVal = REST_HasMarkedChildren (info);

1615 2    if (retVal != BOOL_TRUE)
1616 3    {
1617 3      /* If there are no marks, then nothing is partially marked */
1618 3      LBOX_GoHome (REST_RestoreWin->SelectedListBox);
1619 3      nextInfo = LBOX_CurGetClientData(
                                        REST_RestoreWin->SelectedListBox);
1620 3      while (!retVal && (nextInfo != NULL))
1621 4      {
1622 4        fullName = esl_strdup (REST_GetFullName(nextInfo));
1623 4        childInfo = REST_FindInfoInChildren (
                                            fullName, info, BOOL_FALSE);
1624 4        if ((childInfo != NULL) && (childInfo->marked))
1625 5        {
1626 5          retVal = BOOL_TRUE;
1627 4        }
1628 4        else
1629 5        {
1630 5          LBOX_GoDown (REST_RestoreWin->SelectedListBox);
1631 5          nextInfo = LBOX_CurGetClientData(
                                        REST_RestoreWin->SelectedListBox);
1632 4        }
1633 4        GUTIL_Free (fullName);
1634 3      }
1635 2    }
1636 1  }

1638 1  return (retVal);
1639  }
```

```
1641  /*****************************************************************
1642   * REST_ToggleMarkNfy
1643   *
1644   * Description:
1645   *   This routine will toggle the mark status of the given object.
1646   *
1647   * Parameters:
1648   *   fileObject (I) - The file manager object to toggle
1649   *
1650   * Returns:
1651   *   None.
1652   *
1653   *****************************************************************/

1655  void REST_ToggleMarkNfy ( GFMGR_Context fMgr,
1656                            GFMGR_Object  fileObject )
1657 1 {
1658 1   RestoreInfoPtr   info;
                                    /* The real info for the object */
1659 1   long             numberBad = 0;
                                  /* Number of bad files marked */
1660 1   long             numberMarked = 0;    /* Number of objects marked */
1661 1   RestoreInfoPtr   tmpObject;
                                  /* Pointer to walk the list with */
1662 1   RestoreInfoPtr   newWorkItem = NULL;
                                 /* Work item for the selected object */
1663 1   RestoreInfoPtr   topInfo;
                                /* The parent object in the SArea */

1665 1   /* If a restore is in progress, no marking is allowed! */
1666 1   if (REST_RestoreInProgress() || REST_SearchInProgress())
1667 1     return;

1669 1   /* Cast back to the real object */
1670 1   info = (RestoreInfoPtr)fileObject;

1672 1   if ((info != NULL) &&
1673 1       ((currentWorkItemInfo != NULL) || (
                                        info->type == REST_WorkItem)))
1674 2   {
1675 2     /* Find the work-item object for this object */
1676 2     tmpObject = info;
1677 2     while ((newWorkItem == NULL) && (tmpObject != NULL))
1678 3     {
1679 3       /* If this is the work-item object we're done */
1680 3       if (tmpObject->type == REST_WorkItem)
1681 4       {
1682 4         newWorkItem = tmpObject;
1683 3       }
1684 3       /* Else try its parent */
1685 3       else
1686 4       {
1687 4         tmpObject = tmpObject->parent;
1688 3       }
1689 2     }

1691 2     /* If this is a different workitem */
1692 2     if (newWorkItem != currentWorkItemInfo)
1693 3     {
1694 3       /* If the client is selected, then we might be able to mark */
1695 3       topInfo = (RestoreInfoPtr) GFMGR_GetFirstSelectedObject (
                                                        fileMgrContext);
1696 3       if ((topInfo->type == REST_Client) &&
1697 3           (info->type == REST_WorkItem) &&
1698 3           (REST_VerifySelection (fileMgrContext,
```



```
1699 3                                     newWorkItem,
1700 3                                     BOOL_FALSE,
1701 3                                     BOOL_FALSE)))
1702 4       {
1703 4         /*
1704 4          * Select the work item,
                                  this causes it to be selected in the SArea
1705 4          * which isn't really the desired result,
                                                  but we have to have the
1706 4          * workitem in the SArea in order to continue
1707 4          */

1709 4         GFMGR_SelectObject (fileMgrContext, (
                                                GFMGR_Object)info, BOOL_TRUE);
1710 3       }
1711 3       else
1712 4       {
1713 4         return;
1714 3       }
1715 2     }

1717 2     /* This may take a while so display a wait cursor */
1718 2     WGT_UseWaitCursor ((WgtPtr)REST_RestoreWin);

1720 2     /*
1721 2      * Continue with the operation if:
1722 2      *  The object is marked (anything can be unmarked)
1723 2      *  The object is a higher level object (like a workitem)
1724 2      *  The object is markable for the given time and passes the bad
                                                                  file test
1725 2      */

1727 2     if ((info->marked) ||
1728 2         ((info->type != REST_File) && (
                                          info->type != REST_Directory)) ||
1729 2         (GREST_IsObjectMarkable (GREST_Handle, info->restoreObject) &&
1730 2          ((info->status != Backup_Bad) || REST_MarkBadFiles)))
1731 3     {
1732 3       if ((info->marked && REST_UnmarkInfo (
                                       info, &numberMarked, &numberBad)) ||
1733 3            (!info->marked && REST_MarkInfo (
                                         info, &numberMarked, &numberBad)))
1734 4       {
1735 4         REST_UpdateMarkedDataFromList (numberMarked, numberBad);
1736 4         SAREA_RedrawParts ((SAreaPtr)REST_RestoreWin->SelectedListBox);
1737 4         REST_UpdateRemoveButtons ();
1738 3       }
1739 2     }

1741 2     /* Display the normal cursor again */
1742 2     WGT_UseCursor ((WgtPtr)REST_RestoreWin, CURS_Arrow());
1743 1   }
1744  }
```

```
1746  /*********************************************************************
1747   * REST_FileInitialize
1748   *
1749   * Description:
1750   *   This routine will initialize the routines in the file manager
                                                                    portion
1751   *   of the restore functionality.
1752   *
1753   * Parameters:
1754   *   None.
1755   *
1756   * Returns:
1757   *   None.
1758   *
1759   *********************************************************************/

1761  void REST_FileInitialize (void)
1762 1 {

1764 1   /* Create the file manager context for restore */
1765 1   fileMgrContext = GFMGR_Create
1766 1                    (REST_RestoreWin->BackupSArea,
1767 1                     REST_RestoreWin->BackupListBox,
1768 1                     REST_RestoreWin->LboxVSb,
1769 1                     (TEdPtr)REST_RestoreWin->JunkTEd,
1770 1                     NUMBER_ITEMS_COLUMNS,
1771 1                     BOOL_FALSE,
1772 1                     REST_GetChildren,
1773 1                     REST_CloseChildren,
1774 1                     REST_GetIcon,
1775 1                     REST_GetOverlayIcon,
1776 1                     REST_GetIcon2,
1777 1                     REST_GetOverlayIcon2,
1778 1                     REST_GetNameString,
1779 1                     REST_HasChildren,
1780 1                     REST_HasChildren,
1781 1                     REST_GetBackupCellString,
1782 1                     REST_VerifySelection,
1783 1                     REST_FileSelectionNfy,
1784 1                     REST_LBoxSelectionNfy,
1785 1                     REST_IsMarkable,
1786 1                     REST_IsMarked,
1787 1                     REST_ToggleMarkNfy,
1788 1                     REST_IsPartialMarked);

1790  }
```

```
/*****************************************************************
 * restProgressMgr.c
 *
 *
 * Copyright 1999 by EMC Corp.
 *
 *
 * Mission Statement:
 *   This file contains the functions necessary for the EDM Restore
 *   Progress window.
 *
 * Required includes:
 *   None
 *
 * Compile-Time Options:
 *   N/A
 *
 *
 * RCS Information:
 *     $RCSfile$
 *     $Revision$
 *     $Date$
 *****************************************************************/

#define  ERR_LIB          RESTORE

#include <esl/c_portable.h>
#include <esl/ep_xopen.h>

#include <respub.h>
#include <cboxpub.h>
#include <panelpub.h>
#include <tbutpub.h>
#include <winpub.h>
#include <drawpub.h>

#include "eerrno/e_errno.h"
#include "util/esl_string.h"
#include <restore/restore_api.h>
#include <restore/restore_engine.h>

#include "gutil/timeUtils.h"
#include "gutil/alertMgr.h"
#include "gutil/iconDefs.h"
#include "gutil/iconUtils.h"
#include "gutil/winUtils.h"
#include "gutil/lboxUtils.h"
#include "gutil/guiUtils.h"
#include "gutil/panel.h"

#include "restoreP.h"
#include "restUtils.h"
#include "restCBMgr.h"
#include "restSearch.h"
#include "restFileMgr.h"
#include "restQuestion.h"
#include "restProgress.h"

/*************
 * Constants *
 *************/

#define KBYTES  (1 << 10)
#define MBYTES  (1 << 20)
```



```
/* All delays are in milli-seconds */
#define RESTORE_DONE_DELAY             1
#define RESTORE_CHECK_CANCEL_DELAY 5000
#define RESTORE_CHECK_SUBMIT_DELAY  500
#define RESTORE_CHECK_STARTUP_DELAY 500

#define HORIZONTAL_MARGIN  5
#define VERTICAL_MARGIN    2

#define SEPARATOR_CHAR  '='

/************************
 * Local Data structures *
 ************************/

/********************
 * Global Variables *
 ********************/

static BoolEnum REST_ProgressDisplayed = BOOL_FALSE;
static RestProgressWinPtr REST_ProgressWindow = NULL;

/* Flags for restoral progrss */
static BoolEnum restoreInProgress = BOOL_FALSE;
static BoolEnum restoreCancelled = BOOL_FALSE;

#define NUM_WAIT_ICONS       5
#define UPDATE_ICON_COUNT    1
static IconPtr  waitIcon[NUM_WAIT_ICONS];
static Int      iconNum = 0;
static Int      iconCount = 0;

static Int      checkForCancelId = 0;

static time_t  REST_FirstDataTime = 0;

/*****************************************************************
 * REST_SetRestoreVisibility
 *
 * Description:
 *   This routine will set the visibility status of widgets based on
 *   whether or not a restore is in progress.
 *
 * Parameters:
 *   visible (I) - Value of the visibility to set
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void  REST_SetRestoreVisibility (BoolEnum visible)
{
  BoolEnum origStatus; /* Original status of reread flag */

  /* If the state is not visible,
                              disable options not valid during restore */
  if (!visible)
  {
    /* Restrict all actions in the FS Options Panel */
    GUTIL_WGT_SetEnabled ((
                  WgtPtr)REST_RestoreWin->FSOptionsPanel, BOOL_FALSE);

    /* Restrict marking, unmarking, and searching */
    GUTIL_WGT_SetEnabled ((
                        WgtPtr)REST_RestoreWin->MarkButton, BOOL_FALSE);
```

```
130  2      GUTIL_WGT_SetEnabled ((
                        WgtPtr)REST_RestoreWin->UnmarkButton, BOOL_FALSE);
131  2      GUTIL_WGT_SetEnabled ((
                        WgtPtr)REST_RestoreWin->SearchButton, BOOL_FALSE);


133  2      /* Restrict unmarking from the mark summary */
134  2      GUTIL_WGT_SetEnabled ((
                        WgtPtr)REST_RestoreWin->RemoveButton, BOOL_FALSE);
135  2      GUTIL_WGT_SetEnabled ((
                         WgtPtr)REST_RestoreWin->ClearButton, BOOL_FALSE);


137  2      /* Don't let another restore start */
138  2      GUTIL_WGT_SetEnabled ((
                         WgtPtr)REST_RestoreWin->StartButton, BOOL_FALSE);


140  2      /* Don't let the user close the window */
141  2      GUTIL_WGT_SetEnabled ((
                         WgtPtr)REST_RestoreWin->CloseButton, BOOL_FALSE);


143  2      /* Don't let the user change the path */
144  2      GUTIL_WGT_SetEnabled ((
                           WgtPtr)REST_RestoreWin->PathTEd, BOOL_FALSE);


146  2      /* Don't let the user do anything in the search window */
147  2      if (REST_SearchWindowIsDisplayed ())
148  3      {
149  3        GUTIL_WGT_SetEnabled((
                      WgtPtr)REST_SearchWindow->MarkButton,   BOOL_FALSE);
150  3        GUTIL_WGT_SetEnabled((
                      WgtPtr)REST_SearchWindow->UnMarkButton, BOOL_FALSE);
151  3        GUTIL_WGT_SetEnabled((
                      WgtPtr)REST_SearchWindow->SetViewButton,BOOL_FALSE);
152  2      }

154  1    }

156  1    /* Else, set the states back to what they should be */
157  1    else
158  2    {
159  2      /* If we are doing FS restore,
                                    re-enable FS panel and search button */
160  2      if (currentWorkItemInfo != NULL)
161  3      {
162  3        GUTIL_WGT_SetEnabled ((
                       WgtPtr)REST_RestoreWin->FSOptionsPanel, BOOL_TRUE);
163  3        GUTIL_WGT_SetEnabled ((
                         WgtPtr)REST_RestoreWin->SearchButton, BOOL_TRUE);

165  3        /*
166  3         * Sensitize the date buttons appropriately
167  3         * Temporarily make sure the reread flag is set so no error is
                                                                    displayed
168  3         */

170  3        origStatus = REST_IsReReadInProgress ();
171  3        REST_SetReReadInProgress (BOOL_TRUE);
172  3        REST_UpdateDateButtons ();
173  3        REST_SetReReadInProgress (origStatus);
174  2      }

176  2      /* Reset the enabling of the mark an unmark buttons */
177  2      REST_LBoxSelectionNfy (NULL);

179  2      /* Reset unmarking from the mark summary */
180  2      REST_UpdateRemoveButtons ();
```



```
182  2      /* Reset the start button */
183  2      GUTIL_WGT_SetEnabled ((
                         WgtPtr)REST_RestoreWin->StartButton, BOOL_TRUE);


185  2      /* Reset the Close button */
186  2      GUTIL_WGT_SetEnabled ((
                         WgtPtr)REST_RestoreWin->CloseButton, BOOL_TRUE);


188  2      /* Reset the Path Ted */
189  2      GUTIL_WGT_SetEnabled ((WgtPtr)REST_RestoreWin->PathTEd, BOOL_TRUE);


191  2      /* Reset the search window buttons */
192  2      if (REST_SearchWindowIsDisplayed ())
193  3      {
194  3        REST_SearchUpdateButtons ();
195  2      }
196  1    }
197    }
```

```
199   /*****************************************************************
200    * REST_ProgressDrawPercentComplete
201    *
202    * Description:
203    *   This routine will draw a graph of % complete in the given drawing
204    *   area.
205    *
206    * Parameters:
207    *   drawArea   (I) - The widget to draw to
208    *
209    * Returns:
210    *   None.
211    *
212    *****************************************************************/

214   void  REST_ProgressDrawPercentComplete (WgtPtr drawArea)
215 1 {
216 1   int        percentage;
217 1   Rect16Ptr  wgtBox;
218 1   Rect16Rec  drawBox;
219 1   Rect16Rec  fillBox;
220 1   Char       drawString[256];
221 1   Point16Rec textOri;
222 1   Point16Rec textExt;
223 1   Point16Rec wgtOri;
224 1   Point16Rec wgtExt;
225 1   Point16Rec penExt;
226 1   Rect16Rec  fgBox;
227 1   Rect16Rec  bgBox;
228 1   ColorPtr   fgColor;
229 1   ColorPtr   bgColor;
230 1   eerrno_ty  status;
231 1   Point16Rec iconExt;
232 1   Rect16Rec  iconRect;

234 1   if (drawArea == NULL)
235 1     return;

237 1   percentage = (int)WGT_GetClientRes (drawArea);

239 1   /*
240 1    * We never want to show 100% complete. Let it finish
241 1    */

243 1   if (percentage >= 100)
244 2   {
245 2     percentage = 99;
246 1   }

248 1   /* Set the clip rectangle to the widget */
249 1   if (DRAW_ClipWgt (drawArea))
250 2   {

252 2     /* Get the size of the pen,
                          we only want to draw inside the widget's pen */
253 2     PEN_QuerySize (WGT_GetPen (drawArea), &penExt);

255 2     /* Get the box for the entire widget */
256 2     wgtBox = (Rect16Ptr) WGT_GetBox (drawArea);

258 2     /* Get the origin and extents to draw into (
                                              not overdrawing the pen) */
259 2     wgtOri.x = penExt.x;
260 2     wgtOri.y = penExt.y;
261 2     wgtExt.x = wgtBox->Ext.x - (2 * penExt.x);
```



```
262 2     wgtExt.y = wgtBox->Ext.y - (2 * penExt.y);

264 2     /* Define the rectangle that borders the widget */
265 2     drawBox.Ori.x = wgtOri.x;
266 2     drawBox.Ori.y = wgtOri.y;
267 2     drawBox.Ext.x = wgtExt.x;
268 2     drawBox.Ext.y = wgtExt.y;

270 2     if (restoreInProgress && !restoreCancelled)
271 3     {
272 3       fgColor = COLOR_Transparent ();
273 3       bgColor = WGT_GetBgColor ((WgtPtr)drawArea);
274 2     }
275 2     else
276 3     {
277 3       status = (eerrno_ty)WGT_GetClientRes (drawArea);
278 3       if (restoreCancelled)
279 4       {
280 4         fgColor = WGT_GetFgColor ((WgtPtr)drawArea);
281 4         bgColor = WGT_GetBgColor ((WgtPtr)drawArea);
282 3       }
283 3       else if (status == E_SUCCESS)
284 4       {
285 4         fgColor = COLOR_Black ();
286 4         bgColor = COLOR_Green ();
287 3       }
288 3       else
289 4       {
290 4         fgColor = COLOR_Black ();
291 4         bgColor = COLOR_Red ();
292 3       }
293 2     }

295 2     /* First draw a box around the entire widget */
296 2     DRAW_SetContext (drawArea,
297 2                      COLOR_Transparent (),
298 2                      bgColor,
299 2                      PEN_Solid(),
300 2                      PATT_Empty(),
301 2                      WGT_GetFont (drawArea));
302 2     DRAW_Rect (drawArea, &drawBox);

304 2     if (restoreInProgress && !restoreCancelled)
305 3     {
306 3       if (percentage >= 0)
307 4       {
308 4         /* Now determine how much to fill in */
309 4         fillBox.Ori.x = wgtOri.x;
310 4         fillBox.Ori.y = wgtOri.y;
311 4         fillBox.Ext.x = (Int16)((float)wgtExt.x * ((
                                                    float)percentage/100.0));
312 4         fillBox.Ext.y = wgtExt.y;

314 4         /* Now draw the filled area depicting the percent complete */
315 4         DRAW_SetContext (drawArea,
316 4                          WGT_GetFgColor (drawArea),
317 4                          WGT_GetFgColor (drawArea),
318 4                          PEN_Solid(),
319 4                          PATT_Empty(),
320 4                          WGT_GetFont (drawArea));
321 4         DRAW_Rect (drawArea, &fillBox);
322 3       }

324 3       /* Determine the string to draw and its size */
325 3       if (percentage >= 0)
326 4       {
```

```
327 4           STR_Sprintf (drawString,
328 4                        REST_GetErrorString (REST_PERCENT_PROGRESS),
329 4                        percentage);
330 3         }
331 3         else
332 4         {
333 4           STR_Sprintf (drawString, REST_GetErrorString (
                                                 REST_STARTUP_IN_PROGRESS));
334 3         }

336 3         DRAW_QueryTextExt (drawArea, drawString, &textExt);

338 3         /* Set the origin of the text so that is it displayed in the
                                                                    center */
339 3         textOri.x = (wgtExt.x - textExt.x) / 2;
340 3         textOri.y = penExt.y + ((wgtExt.y - textExt.y) / 2);

342 3         /* First draw the string in the foreground color */
343 3         fgBox.Ori.x = textOri.x;
344 3         fgBox.Ori.y = textOri.y;
345 3         fgBox.Ext.x = textExt.x;
346 3         fgBox.Ext.y = textExt.y;

348 3         DRAW_SetContext (drawArea,
349 3                          WGT_GetFgColor (drawArea),
350 3                          WGT_GetBgColor (drawArea),
351 3                          PEN_Solid(),
352 3                          PATT_Empty(),
353 3                          WGT_GetFont (drawArea));
354 3         DRAW_ClippedText (drawArea,
355 3                           &fgBox,
356 3                           DRAW_JUSTLEFT | DRAW_JUSTVCENTER,
357 3                           drawString)

359 3         /*
360 3          * If the filled area overlaps the text area, we need to draw
361 3          * the portion that overlaps in the background color
362 3          */

364 3         if ((percentage >= 0) && (textOri.x < fillBox.Ext.x))
365 4         {
366 4           /* Set the background box to be the entire string */
367 4           bgBox.Ori.x = textOri.x;
368 4           bgBox.Ori.y = textOri.y;
369 4           bgBox.Ext.x = textExt.x;
370 4           bgBox.Ext.y = textExt.y;

372 4           /*
373 4            * If the filled area ends before the end of the string,
                                                                  change the
374 4            * box to end at the end of the filled area
375 4            */

377 4           if (bgBox.Ext.x > ((
                                fillBox.Ori.x + fillBox.Ext.x) - textOri.x))
378 4             bgBox.Ext.x = (fillBox.Ori.x + fillBox.Ext.x) - textOri.x;

380 4           DRAW_SetContext (drawArea,
381 4                            WGT_GetBgColor (drawArea),
382 4                            WGT_GetFgColor (drawArea),
383 4                            PEN_Solid(),
384 4                            PATT_Empty(),
385 4                            WGT_GetFont (drawArea));
386 4           DRAW_ClippedText (drawArea,
387 4                             &bgBox,
388 4                             DRAW_JUSTLEFT | DRAW_JUSTVCENTER,
```



```
389 4                            drawString)
390 3         }
391 3         else if (percentage < 0)
392 4         {
393 4           /* draw the icon */
394 4           ICON_QuerySize (waitIcon[iconNum], &iconExt);

396 4           /* This defines the box in which we can place the icons so far
                                                                          */
397 4           iconRect.Ori.x = fgBox.Ori.x + fgBox.Ext.x + HORIZONTAL_MARGIN;
398 4           iconRect.Ori.y = (wgtExt.y - iconExt.y) / 2;
399 4           iconRect.Ext.x = iconExt.x;
400 4           iconRect.Ext.y = iconExt.y;

402 4           DRAW_Icon (drawArea, &iconRect, waitIcon[iconNum]);

404 4           iconCount++;
405 4           if (iconCount == UPDATE_ICON_COUNT)
406 5           {
407 5             iconCount = 0;
408 5             iconNum++;
409 5             if (iconNum == NUM_WAIT_ICONS)
410 6             {
411 6               iconNum = 0;
412 5             }
413 4           }
414 3         }
415 2       }
416 2       else
417 3       {
418 3         status = (eerrno_ty)WGT_GetClientRes (drawArea);

420 3         if (restoreCancelled)
421 4         {
422 4           STR_Cpy (drawString, REST_GetErrorString (REST_CANCEL_TITLE));
423 3         }
424 3         else if (status == E_SUCCESS)
425 4         {
426 4           STR_Cpy (drawString, REST_GetErrorString (
                                                   REST_RESTORE_SUCCESS));
427 3         }
428 3         else
429 4         {
430 4           STR_Cpy (drawString, REST_GetErrorString (
                                                   REST_RESTORE_FAILURE));
431 3         }

433 3         DRAW_QueryTextExt (drawArea, drawString, &textExt);

435 3         /* Set the origin of the text so that is it displayed in the
                                                                    center */
436 3         textOri.x = (wgtExt.x - textExt.x) / 2;
437 3         textOri.y = penExt.y + ((wgtExt.y - textExt.y) / 2);

439 3         /* First draw the string in the foreground color */
440 3         fgBox.Ori.x = textOri.x;
441 3         fgBox.Ori.y = textOri.y;
442 3         fgBox.Ext.x = textExt.x;
443 3         fgBox.Ext.y = textExt.y;

445 3         DRAW_SetContext (drawArea,
446 3                          fgColor,
447 3                          bgColor,
448 3                          PEN_Solid(),
449 3                          PATT_Empty(),
450 3                          WGT_GetFont (drawArea));
```

```
451  3          DRAW_ClippedText (drawArea,
452  3                            &fgBox,
453  3                            DRAW_JUSTLEFT | DRAW_JUSTVCENTER,
454  3                            drawString)

456  2       }

458  2       /* End clipping */
459  2       DRAW_ClipEnd (drawArea);

461  1     }
462       }
```



```
464   /*****************************************************************
465    * Function Name:   REST_GetDataSizeString()
466    *
467    * Description:
468    *   This function will set the give string to represent the value
469    *   of the number of kbytes given in sizeInKB.
470    *
471    *   If the value is less than 1000,        it will display in KB
472    *   If the value is greater than 1000,     it will display in MB
473    *   If the value is greater than 1000000, it will display in GB
474    *
475    * Parameters:
476    *   (I) dataSize - Size of the data in KB
477    *   (O) sizeString - PREALLOCATED string to put size value into
478    *
479    * Success Outputs and Side Effects:
480    *   None.
481    *
482    *****************************************************************/
483   void REST_GetDataSizeString (u_long sizeInKB,
484                                Str    sizeString)
485 1 {

487 1     if (sizeInKB < KBYTES)
488 2     {
489 2       /* Display K Bytes */
490 2       STR_Sprintf (sizeString, "%lu KB", sizeInKB);
491 1     }
492 1     else if (sizeInKB < MBYTES)
493 2     {
494 2       /* Display M Bytes with 2 decimal places */
495 2       STR_Sprintf (sizeString, "%.2f MB",
496 2                    (float)sizeInKB / (float)KBYTES);
497 1     }
498 1     else
499 2     {
500 2       /* Display G Bytes with 2 decimal places */
501 2       STR_Sprintf (sizeString, "%.2f GB",
502 2                    (float)sizeInKB / (float)MBYTES);
503 1     }
504   }
```

```
506  /*****************************************************************
507   * REST_ProgressUpdate
508   *
509   * Description:
510   *   This routine will display the Progress window.
511   *
512   * Parameters:
513   *   currentProgress - The progress information to display
514   *
515   * Returns:
516   *   BOOL_TRUE  - if the user hit the cancel button
517   *   BOOL_FALSE - otherwise.
518   *
519   *****************************************************************/

521  static void  REST_ProgressUpdate (feedbackObjectPtr currentProgress)
522 1{
523 1  Int32           kbSoFar = 0;
524 1  Int32           expectedKB = 0;
525 1  Int32           filesSoFar = 0;
526 1  Int32           expectedFiles = 0;
527 1  EDMProgressPtr  objectPtr;
528 1  Char            sizeDone[16];
529 1  Char            sizeTotal[16];
530 1  Char            outString[64];
531 1  time_t          startTime;
532 1  time_t          currentTime;
533 1  time_t          timeRunning;
534 1  time_t          eta;
535 1  float           percentage;

537 1  /* If the progress window is not displayed, we're done */
538 1  if (!REST_ProgressDisplayed)
539 1    return;

541 1  if (currentProgress != NULL)
542 2  {
543 2    /*
544 2     * Tally up the totals
545 2     */

547 2    objectPtr = EDMRST_GetFirstEDMObject (
                                        GREST_Handle, currentProgress);
548 2    while (objectPtr != NULL)
549 3    {
550 3      kbSoFar += EDMRST_GetObjectEDMTotalKbytesSofar (
                                        GREST_Handle, objectPtr);
551 3      expectedKB += EDMRST_GetObjectEDMTotalKBExpected (
                                        GREST_Handle, objectPtr);
552 3      filesSoFar += EDMRST_GetObjectEDMTotalFiles (
                                        GREST_Handle, objectPtr);
553 3      expectedFiles+= EDMRST_GetObjectEDMTotalFilesExpected (
                                                    GREST_Handle,
554 3
                                                      objectPtr);

556 3      startTime = EDMRST_GetObjectEDMTimeStarted (
                                        GREST_Handle, objectPtr);
557 3      currentTime = EDMRST_GetObjectEDMCurrTime (
                                        GREST_Handle, objectPtr);

559 3      objectPtr = EDMRST_GetNextEDMObject (GREST_Handle, objectPtr);
560 2    }

562 2    TED_SetStr ((TEdPtr)REST_ProgressWindow->WiNameTEd,
```



```
563 2                 currentWorkItemInfo->name);

565 2    REST_GetDataSizeString (kbSoFar, sizeDone);
566 2    REST_GetDataSizeString (expectedKB, sizeTotal);

568 2    TED_SetStrf ((TEdPtr)REST_ProgressWindow->StatusTed,
569 2                 REST_GetErrorString (REST_DATA_SIZE_PROGRESS),
570 2                 sizeDone,
571 2                 sizeTotal);
572 2    TED_InsertStr ((TEdPtr)REST_ProgressWindow->StatusTed, "\n\n");
573 2    TED_InsertStrf ((TEdPtr)REST_ProgressWindow->StatusTed,
574 2                    REST_GetErrorString (REST_DATA_FILE_PROGRESS),
575 2                    filesSoFar,
576 2                    expectedFiles);

578 2    if ((kbSoFar > 0) || (filesSoFar > 0))
579 3    {
580 3      /* See if this is the first tick */
581 3      if (REST_FirstDataTime == 0)
582 3        REST_FirstDataTime = currentTime - 1;

584 3      timeRunning = currentTime - REST_FirstDataTime;

586 3      if (expectedKB > 0)
587 4      {
588 4        percentage = (float)kbSoFar / (float)expectedKB;
589 4        if (percentage == 0)
590 4          percentage = (float)1 / (float)1000;

592 3      }
593 3      else
594 4      {
595 4        percentage = (float)1 / (float)1000;
596 3      }

598 3      WGT_SetClientRes ((WgtPtr)REST_ProgressWindow->ProgressArea,
599 3                        (ClientPtr)((int)((100 * percentage) + 0.5)));

601 3      eta = REST_FirstDataTime + (int)((
                                          float)timeRunning / percentage);

603 3      strftime (outString,
604 3                MEDIUM_STRING_LENGTH,
605 3                "%H:%M",
606 3                localtime (&startTime));
607 3      TED_SetStr ((
                     TEdPtr)REST_ProgressWindow->StartTimeTEd, outString);

609 3      strftime (outString,
610 3                MEDIUM_STRING_LENGTH,
611 3                "%H:%M",
612 3                localtime (&eta));
613 3      TED_SetStr ((TEdPtr)REST_ProgressWindow->EndTimeTEd, outString);
614 2    }
615 1  }

617 1  if ((currentProgress == NULL) || ((kbSoFar == 0) && (
                                                    filesSoFar == 0)))
618 2  {
619 2    TED_SetStr ((TEdPtr)REST_ProgressWindow->StartTimeTEd,      "");
620 2    TED_SetStr ((TEdPtr)REST_ProgressWindow->EndTimeTEd,        "");
621 2    WGT_SetClientRes ((WgtPtr)REST_ProgressWindow->ProgressArea, (
                                                          ClientPtr)-1);
622 2    REST_FirstDataTime = 0;
623 1  }
```

```
625 1   REST_ProgressDrawPercentComplete ((
                                WgtPtr)REST_ProgressWindow->ProgressArea);
626     }
```



```
628     /*************************************************************************
629      * REST_ProgressDisplay
630      *
631      * Description:
632      *   This routine will display the Progress window.
633      *
634      * Parameters:
635      *   None
636      *
637      * Returns:
638      *   None.
639      *
640      *************************************************************************/

642     void REST_ProgressDisplay ( void )
643   1 {
644   1   if (REST_ProgressDisplayed)
645   2   {
646   2     REST_ProgressRemove ();
647   1   }

649   1   /* Load up the window's resources */
650   1   REST_ProgressWindow = REST_ProgressWindowInit ((
                                                    WinPtr)REST_RestoreWin);

652   1   /* Flag the progress window as displayed */
653   1   REST_ProgressDisplayed = BOOL_TRUE;

655   1   /* Add the host to the window title */
656   1   GUTIL_AddHostToWindowTitle ((WinPtr)REST_ProgressWindow);

658   1   TED_SetStr ((TEdPtr)REST_ProgressWindow->WiNameTEd,
659   1               currentWorkItemInfo->name);

661   1   WGT_SetClientRes ((WgtPtr)REST_ProgressWindow->ProgressArea, (
                                                              ClientPtr)-1);

663   1   waitIcon[0] = (IconPtr) RES_Load ("Icons", "WaitIcon");
664   1   waitIcon[1] = (IconPtr) RES_Load ("Icons", "Wait2Icon");
665   1   waitIcon[2] = (IconPtr) RES_Load ("Icons", "Wait3Icon");
666   1   waitIcon[3] = (IconPtr) RES_Load ("Icons", "Wait4Icon");
667   1   waitIcon[4] = (IconPtr) RES_Load ("Icons", "Wait5Icon");
668   1   iconNum = 0;
669   1   iconCount = 0;

671   1   /* Put up the window */
672   1   WIN_Show ((WinPtr)REST_ProgressWindow);
673     }
```

```
675  /*****************************************************************
676   * REST_ProgressDisplayError
677   *
678   * Description:
679   *   This routine will display an error in the  Progress window.
680   *
681   * Parameters:
682   *   status          - The error number
683   *   feedbackObject  - The feedback object for more info
684   *
685   * Returns:
686   *   None.
687   *
688   *****************************************************************/

690  static void REST_ProgressDisplayError (eerrno_ty          status,
691                                         feedbackObjectPtr
                                                          feedbackObject)
692  1 {
693  1   Char            drawString[256];
694  1   Rect16Rec       wgtRect;
695  1   Rect16Rec       textRect;
696  1   Rect16Ptr       origBox;
697  1   Rect16Ptr       winBox;
698  1   EDMProgressPtr edmObject;

700  1   if (REST_ProgressDisplayed)
701  2   {
702  2     if (status != E_SUCCESS)
703  3     {
704  3       if ((feedbackObject != NULL) &&
705  3           ((edmObject = EDMRST_GetFirstEDMObject (GREST_Handle,
706  3                                                   feedbackObject)) !=
                                                                 NULL))

707  4       {
708  4         TED_SetStrf ((TEdPtr)REST_ProgressWindow->StatusTed,
709  4                      "\n%s\n",
710  4                      e_get_error_text(status));
711  3       }
712  3       else
713  4       {
714  4         TED_SetStr ((TEdPtr)REST_ProgressWindow->StatusTed,
715  4                     e_get_error_text(status));
716  3       }
717  2     }

719  2     if (status == E_SUCCESS)
720  3     {
721  3       WIN_SetLabel ((WinPtr)REST_ProgressWindow,
722  3                     REST_GetErrorString (REST_SUCCESS_INDEX));
723  3       STR_Cpy (drawString, REST_GetErrorString (REST_RESTORE_SUCCESS));
724  2     }
725  2     else
726  3     {
727  3       WIN_SetLabel ((WinPtr)REST_ProgressWindow,
728  3                     REST_GetErrorString (REST_RESTORE_FAILURE));
729  3       STR_Cpy (drawString, e_get_error_text(status));
730  2     }

732  2     GUTIL_AddHostToWindowTitle ((WinPtr)REST_ProgressWindow);

734  2     /* Set the error as the client data */
735  2     WGT_SetClientRes ((WgtPtr)REST_ProgressWindow->ProgressArea,
736  2                       (ClientPtr)status);
```



```
738  2     WGT_Inval ((WgtPtr)REST_ProgressWindow->ProgressArea, BOOL_FALSE);

740  2     GUTIL_WGT_SetEnabled ((
                        WgtPtr)REST_ProgressWindow->CancelTBut, BOOL_TRUE);
741  2     TBUT_SetLabel ((TButPtr)REST_ProgressWindow->CancelTBut, "&OK");
742  1   }
743  1   else
744  2   {
745  2     if (status == E_SUCCESS)
746  3     {
747  3       /* Tell the user that the restoral completed successfully */
748  3       GALERT_DisplayError ((WinPtr)REST_RestoreWin,
749  3                            REST_GetErrorString (REST_SUCCESS_INDEX),
750  3                            GICON_GetIcon (I_SUCCESS),
751  3                            REST_GetErrorString (REST_RESTORE_SUCCESS));
752  2     }
753  2     else
754  3     {
755  3       /* Display the error */
756  3       GALERT_DisplayError ((WinPtr)REST_RestoreWin,
757  3                            REST_GetErrorString (REST_RESTORE_FAILURE),
758  3                            GICON_GetError(),
759  3                            e_get_error_text(status));
760  2     }


763  1   }
764  }
```

```
766  /*********************************************************************
767   * Function Name:   REST_ProgressRemove ()
768   *
769   * Description:
770   *   This routine will remove the progress window.
771   *
772   * Parameters:
773   *   None.
774   *
775   * Success Outputs and Side Effects:
776   *   None.
777   *
778   * Returns:
779   *   None.
780   *
781   *********************************************************************/

783  void REST_ProgressRemove (void)
784 1{
785 1  /* We only care if the window is displayed */
786 1  if (REST_ProgressDisplayed)
787 2  {
788 2    /* Reset the flags */
789 2    REST_ProgressDisplayed = BOOL_FALSE;
790 2    restoreInProgress = BOOL_FALSE;

792 2    /* Remove the window */
793 2    WIN_Terminate ((WinPtr)REST_ProgressWindow);
794 2    REST_ProgressWindow = NULL;
795 1  }
796  }
```



```
798  /*********************************************************************
799   * REST_ProgressCancel
800   *
801   * Description:
802   *   This routine will signal the progress dialog that the cancel
803   *   button was hit.
804   *
805   * Parameters:
806   *   userRequest - Flag if the user requested the cancel
807   *
808   * Returns:
809   *   None.
810   *
811   *********************************************************************/

813  void  REST_ProgressCancel (BoolEnum userRequest)
814 1{
815 1  if (restoreInProgress)
816 2  {
817 2    if (!userRequest ||
818 2        (GALERT_DisplayQuestion((WinPtr)REST_RestoreWin,
819 2                                REST_GetErrorString(
                                            REST_VERIFY_CANCEL_TITLE),
820 2                                GICON_GetQuestion(),
821 2                                REST_GetErrorString(
                                          REST_VERIFY_CANCEL_MESSAGE),
822 2                                BOOL_FALSE) == GALERT_Affirmative))
823 3    {
824 3      /*
825 3       * the user cancelled, display a cancel in progress dialog
826 3       */

828 3      restoreCancelled = BOOL_TRUE;

830 3      WIN_SetLabel ((WinPtr)REST_ProgressWindow,
831 3                    REST_GetErrorString (REST_CANCEL_TITLE));
832 3      GUTIL_AddHostToWindowTitle ((WinPtr)REST_ProgressWindow);

834 3      TED_SetStr ((TEdPtr)REST_ProgressWindow->StatusTed,
835 3                  REST_GetErrorString (REST_CANCEL_PROGRESS));
836 3      WGT_Inval ((
                  WgtPtr)REST_ProgressWindow->ProgressArea, BOOL_FALSE);

838 3      GUTIL_WGT_SetEnabled((
                  WgtPtr)REST_ProgressWindow->CancelTBut, BOOL_FALSE);
839 2    }
840 1  }
841 1  else
842 2  {
843 2    REST_ProgressRemove ();
844 1  }
845  }
```

```
847  /*****************************************************************
848   * REST_DisplayDone
849   *
850   * Description:
851   *   This routine is the routine which will display the done
852   *   dialog with status for a file system restore.
853   *
854   * Parameters:
855   *   status          (I) - The restore status.
856   *   feedbackObject (I) - The restore feedback object
857   *
858   * Returns:
859   *   None.
860   *
861   *****************************************************************/

863  void REST_DisplayDone (eerrno_ty         status,
864                         feedbackObjectPtr feedbackObject)

866  1 {
867  1   EDMProgressPtr edmObject;
868  1   WIProgressPtr  wiObject;
869  1   Char           outString[MEDIUM_STRING_LENGTH];
870  1   time_t         now;

872  1   /* Reset the restore flags */
873  1   restoreInProgress = BOOL_FALSE;
874  1   restoreCancelled = BOOL_FALSE;

876  1   /* Update the end time to the current time */
877  1   now = time((time_t *)NULL);
878  1   strftime (outString,
879  1             MEDIUM_STRING_LENGTH,
880  1             "%H:%M",
881  1             localtime (&now));
882  1   TED_SetStr ((TEdPtr)REST_ProgressWindow->EndTimeTEd, outString);

884  1   /* If the status is successful, check for underlying errors */
885  1   if (status == E_SUCCESS)
886  2   {
887  2     edmObject = EDMRST_GetFirstEDMObject (
                                          GREST_Handle, feedbackObject);
888  2     if (EDMRST_GetObjectEDMFailed (GREST_Handle, edmObject) > 0)
889  3     {
890  3       wiObject = EDMRST_GetFirstObject (GREST_Handle, feedbackObject);
891  3       while ((status == E_SUCCESS) && (wiObject != NULL))
892  4       {
893  4         status = EDMRST_GetObjectItemStatus (GREST_Handle, wiObject);
894  4         wiObject = EDMRST_GetNextObject (GREST_Handle, wiObject);
895  3       }
896  2     }
897  1   }

899  1   /* Display the return status */
900  1   REST_ProgressDisplayError (status, feedbackObject);

902  1   REST_SetRestoreVisibility (BOOL_TRUE);
903  }
```



```
905  /*****************************************************************
906   * REST_RestoreInProgress
907   *
908   * Description:
909   *   This routine returns whether or not a restoral is currently in
                                                              progress.
910   *
911   * Parameters:
912   *   None.
913   *
914   * Returns:
915   *   BOOL_TRUE  - If a restoral has been started (
                                                  and is not yet finished)
916   *   BOOL_FALSE - Otherwise.
917   *
918   *****************************************************************/

920  BoolEnum  REST_RestoreInProgress (void)
921  1 {
922  1   return (restoreInProgress);
923  }
```

```
925  /*****************************************************************
926   * REST_DisplayQuestion
927   *
928   * Description:
929   *   This routine will display a question needed by the running
930   *   restoral to the user.
931   *
932   * Parameters:
933   *   None.
934   *
935   * Returns:
936   *   None.
937   *
938   *****************************************************************/

940  static void REST_DisplayQuestion (void)
941 1 {
942 1   eerrno_ty              status;
943 1   queryObjectPtr         queryObject;
944 1   int                    questionType;
945 1   GPANEL_FieldInfo      *panelFields;
946 1   int                    i;
947 1   long                   cookie = INIT_COOKIE;
948 1   int                    numAnswers = 0;
949 1   Str                   *userAnswers = NULL;
950 1   GPANEL_PanelIndexType  panelType;
951 1   int                    isDefault;
952 1   int                    numChoices;
953 1   int                    minLen;
954 1   GPANEL_PanelHandle     panHandle;
955 1   Char                   tempString[16];
956 1   int                    selections;
957 1   int                    selNumber;
958 1   static BoolEnum        questionDisplayed = BOOL_FALSE;
959 1   Char                   outString[64];

961 1   if (!questionDisplayed)
962 2   {

964 2     if ((status = EDMRST_AllocQueryObject (GREST_Handle,
965 2                                            &queryObject)) !=
                                                                E_SUCCESS)
966 3     {
967 3       REST_DisplayErrnoMessage ((WinPtr)REST_ProgressWindow,
968 3                                 NULL,
969 3                                 NULL,
970 3                                 status);
971 3       return;
972 2     }

974 2     /* Flag that we are displaying a question dialog already */
975 2     questionDisplayed = BOOL_TRUE;

977 2     if (EDMRST_GetQuestion (GREST_Handle, queryObject) == E_SUCCESS)
978 3     {
979 3       questionType = EDMRST_GetQuestionType (
                                            GREST_Handle, queryObject);

981 3       if (questionType == QTYPE_YESNO)
982 4       {
983 4         numAnswers = 1;
984 4         userAnswers = GUTIL_Calloc (sizeof(Str));
985 4         if (GALERT_DisplayQuestion
986 4                 ((WinPtr)REST_RestoreWin,
987 4                  (Str)EDMRST_GetQuestionHeaderText (
```



```
                                                GREST_Handle, queryObject),
988 4                   GICON_GetQuestion(),
989 4                   (Str)EDMRST_GetQuestionText (
                                                GREST_Handle, queryObject),
990 4                   BOOL_FALSE) == GALERT_Affirmative)
991 5         {
992 5           userAnswers[0] = esl_strdup ("Yes");
993 4         }
994 4         else
995 5         {
996 5           userAnswers[0] = esl_strdup ("No");
997 4         }
998 3       }
999 3       else
1000 4      {

1002 4        switch (questionType)
1003 5        {
1004 5          case QTYPE_BOOL:
1005 5            panelType = GPANEL_1RADIO;
1006 5            panHandle = GPANEL_CreateHandle (panelType);
1007 5            panelFields = GPANEL_GetFields (panHandle);
1008 5            panelFields[0].details.radio.numChoices = 2;
1009 5            panelFields[0].details.radio.choices = GUTIL_Calloc (
                                                      2 * sizeof(Str));
1010 5            panelFields[0].details.radio.choices[0] = esl_strdup (
                                                          "True");
1011 5            panelFields[0].details.radio.choices[1] = esl_strdup (
                                                          "False");
1012 5            panelFields[0].details.radio.selection = 0;
1013 5            break;

1015 5          case QTYPE_RAD:
1016 5            panelType = GPANEL_1RADIO;
1017 5            panHandle = GPANEL_CreateHandle (panelType);
1018 5            panelFields = GPANEL_GetFields (panHandle);
1019 5            panelFields[0].details.radio.numChoices =
1020 5                                  EDMRST_GetQuestionNumChoices (
                                                    GREST_Handle,
1021 5                                                  queryObject);

1023 5            /* ONLY ALLOW THE MAXIMUM NUMBER OF RADIO CHOICES */
1024 5            if (
                  panelFields[0].details.radio.numChoices > GPANEL_MAX_RBUT)
1025 6            {
1026 6              printf (
"Unsupported number of choices in radio box: %d. Can only accept %d.
                                                          \n",
1027 6                      panelFields[0].details.radio.numChoices,
1028 6                      GPANEL_MAX_RBUT);
1029 6              panelFields[0].details.radio.
                                          numChoices = GPANEL_MAX_RBUT;
1030 5            }

1032 5            panelFields[0].details.radio.choices =
1033 5               GUTIL_Calloc (
                    panelFields[0].details.radio.numChoices * sizeof(Str));
1034 5            for (i=0; i < panelFields->details.radio.numChoices; i++)
1035 6            {
1036 6              panelFields[0].details.radio.choices[i] =
1037 6                esl_strdup (EDMRST_GetQuestionNextChoice (
                                                    GREST_Handle,
1038 6                                                   queryObject,
1039 6                                                   &isDefault,
1040 6                                                   &cookie));
```

```
                if (isDefault)
                {
                  panelFields[0].details.radio.selection = i;
                }
              }
              break;

            case QTYPE_MULTI:
              numChoices = EDMRST_GetQuestionNumChoices (
                                              GREST_Handle, queryObject);
              if (numChoices > 3)
              {
                printf (
                 "Unsupported number of choices: %d, can only accept 3\n",
                         numChoices);
                numChoices = 3;
              }

              if (numChoices == 1)
              {
                panelType = GPANEL_1TOGGLE;
              }
              else if (numChoices == 2)
              {
                panelType = GPANEL_2TOGGLE;
              }
              else
              {
                panelType = GPANEL_3TOGGLE;
              }

              panHandle = GPANEL_CreateHandle (panelType);
              panelFields = GPANEL_GetFields (panHandle);

              for (i=0; i < numChoices; i++)
              {
                panelFields[i].fieldName = esl_strdup
                         (EDMRST_GetQuestionNextChoice (GREST_Handle,
                                                        queryObject,
                                                        &isDefault,
                                                        &cookie));
                panelFields[i].details.toggle.isSelected = isDefault;
              }
              break;

            case QTYPE_STR:
              panelType = GPANEL_1TED;
              panHandle = GPANEL_CreateHandle (panelType);
              panelFields = GPANEL_GetFields (panHandle);
              panelFields[0].fieldName = NULL;
              panelFields[0].details.ted.isPassword = BOOL_FALSE;
              panelFields[0].details.ted.isOutputOnly = BOOL_FALSE;
              EDMRST_GetQuestionAnswerSize (GREST_Handle,
                                            queryObject,
                                            &minLen,
                                            &panelFields[0].details.ted.
                                                                maxLength);
              panelFields[0].details.ted.invalidChars =
                    esl_strdup (EDMRST_GetQuestionInvalidChars (
                                                          GREST_Handle,
                                                          queryObject));
              panelFields[0].details.ted.value = NULL;
              break;

            case QTYPE_INT:
```



```
              panelType = GPANEL_1SPIN;
              panHandle = GPANEL_CreateHandle (panelType);
              panelFields = GPANEL_GetFields (panHandle);
              panelFields[0].fieldName = NULL;
              panelFields[0].details.spin.value = 0;
              break;

            default:
              STR_Sprintf (outString,
                           "Unsupported question type %d.
                                          \nContact customer support.",
                           questionType);
              GALERT_DisplayError ((WinPtr)REST_RestoreWin,
                                   "Fatal Error",
                                   GICON_GetError(),
                                   outString);
              questionDisplayed = BOOL_FALSE;
              REST_ProgressCancel (BOOL_FALSE);
              EDMRST_FreeQueryObject (GREST_Handle, &queryObject);
              return;
          }

          /* Now create the panel with the appropriate question */
          GPANEL_CreatePanel (panHandle,
                              NULL,
                              (Str)EDMRST_GetQuestionText (GREST_Handle,
                                                           queryObject),
                              NULL);

          /* Create the window to display the panel in */
          REST_QuestionShow ((WinPtr)REST_RestoreWin,
                             queryObject,
                             panHandle,
                             panelFields);

          /* OK, now we have the answers,
                                        so fill in the answer structure */
          switch (questionType)
          {
            case QTYPE_BOOL:
              numAnswers = 1;
              userAnswers = GUTIL_Calloc (sizeof(Str));
              if (panelFields[0].details.radio.selection == 1)
              {
                userAnswers[0] = esl_strdup ("True");
              }
              else
              {
                userAnswers[0] = esl_strdup ("False");
              }
              break;

            case QTYPE_RAD:
              numAnswers = 1;
              userAnswers = GUTIL_Calloc (sizeof(Str));
              userAnswers[0] = esl_strdup (
        panelFields[0].details.radio.choices[panelFields[0].details.radio.
                                                              selection]);
              break;

            case QTYPE_MULTI:
              /* Count the number of selections */
              numAnswers = 0;
              for (i=0; i < numChoices; i++)
              {
                if (panelFields[i].details.toggle.isSelected)
```

```
1164  6                 numAnswers++;
1165  5               }

1167  5               /* If no selections, send NULL as the answer */
1168  5               if (numAnswers == 0)
1169  6               {
1170  6                 numAnswers = 1;
1171  6                 userAnswers = GUTIL_Calloc (sizeof(Str));
1172  6                 userAnswers[0] = NULL;
1173  5               }
1174  5               else
1175  6               {
1176  6                 userAnswers = GUTIL_Calloc (selections * sizeof(Str));
1177  6                 for (i=0; i < numChoices; i++)
1178  7                 {
1179  7                   if (panelFields[i].details.toggle.isSelected)
1180  8                   {
1181  8                     userAnswers[selNumber] = esl_strdup (
                                                    panelFields[i].fieldName);
1182  8                     selNumber++;
1183  7                   }
1184  6                 }
1185  5               }
1186  5               break;

1188  5             case QTYPE_STR:
1189  5               numAnswers = 1;
1190  5               userAnswers = GUTIL_Calloc (sizeof(Str));
1191  5               if (panelFields[0].details.ted.value != NULL)
1192  5                 userAnswers[0] = esl_strdup (
                                       panelFields[0].details.ted.value);
1193  5               else
1194  5                 userAnswers[0] = NULL;
1195  5               break;

1197  5             case QTYPE_INT:
1198  5               numAnswers = 1;
1199  5               userAnswers = GUTIL_Calloc (sizeof(Str));
1200  5               sprintf (
                           tempString, "%d", panelFields[0].details.spin.value);
1201  5               userAnswers[0] = esl_strdup (tempString);
1202  5               break;

1204  5             default:
1205  5               break;
1206  4           }

1208  4           /* Now, clean up the memory for the generic panel */
1209  4           GPANEL_DestroyHandle (&panHandle, BOOL_TRUE);
1210  3         }

1212  3         /* Now give the restore API the answer(s) */
1213  3         for (i=0; i < numAnswers; i++)
1214  4         {
1215  4           EDMRST_SetUserAnswer (GREST_Handle,
1216  4                                 queryObject,
1217  4                                 userAnswers[i],
1218  4                                 i < (numAnswers - 1));

1220  4           /* Done with the string for this answer, so free the memory */
1221  4           GUTIL_Free (userAnswers[i]);
1222  3         }

1224  3         /* Free the memory for the answer array */
1225  3         GUTIL_Free (userAnswers);
1226  2       }
```



```
1228  2       EDMRST_FreeQueryObject (GREST_Handle, &queryObject);

1230  2       /* Flag that the question dialog is no longer displayed */
1231  2       questionDisplayed = BOOL_FALSE;
1232  1     }
1233    }
```

```
1235  /*****************************************************************
1236   * REST_CheckForCancel
1237   *
1238   * Description:
1239   *   This routine will check to see if the restore has been cancelled
1240   *   by the user.
1241   *
1242   * Parameters:
1243   *   None.
1244   *
1245   * Returns:
1246   *   None.
1247   *
1248   *****************************************************************/

1250  void REST_CheckForCancel (ClientPtr clientData)
1251  1 {
1252  1   eerrno_ty              status;
                                                    /* Error Status */

1253  1   feedbackObjectPtr      feedbackObject;
1254  1   static RERunningState lastState = RE_STATE_TIMEOUT;
1255  1   RERunningState         currentState = 0;
1256  1   BoolEnum               done = BOOL_FALSE;
1257  1   NotifyObjectPtr        notifyObject;
1258  1   Str                    nextString;
1259  1   Char                   stateString[GMAX_COMMAND_LENGTH];
1260  1   Char                   separatorString[GMAX_COMMAND_LENGTH];
1261  1   Int                    count;
1262  1   Int                    i;

1264  1   /* We only care if a restore is in progress */
1265  1   if (restoreInProgress &&
1266  1       (REST_ProgressWindow != NULL) &&
1267  1       RES_IsInitialized ((ResPtr)REST_ProgressWindow))

1269  2   {

1271  2     if ((status = EDMRST_AllocFeedbackObject (GREST_Handle,
1272  2                                               &feedbackObject)) !=
                                                              E_SUCCESS)

1273  3     {
1274  3       REST_DisplayErrnoMessage ((WinPtr)REST_ProgressWindow,
1275  3                                 NULL,
1276  3                                 REST_GetErrorString (
                                                    REST_START_ERROR),
1277  3                                 status);
1278  3       return;
1279  2     }

1281  2     status = EDMRST_GetRestoreFeedback (GREST_Handle,
1282  2                                         restoreCancelled,
1283  2                                         &currentState,
1284  2                                         feedbackObject);

1286  2     if (!restoreCancelled)
1287  3     {
1288  3       REST_ProgressUpdate (feedbackObject);
1289  2     }

1291  2     if (status != EP_RB_RECOVER_RPC_INCOMPLETE)
1292  3     {
1293  3       done = BOOL_TRUE;
1294  3       restoreCancelled = BOOL_FALSE;
1295  2     }
```



```
1297  2     if (currentState != lastState)
1298  3     {
1299  3       /* Save the current state */
1300  3       lastState = currentState;

1302  3       /* Get the string to display */
1303  3       STR_Cpy (stateString, EDMRST_GetFeedbackStatus (currentState));

1305  3       /* Create a separator string of the same length */
1306  3       count = STR_Len (stateString);
1307  3       for (i=0; i < count; i++)
1308  4       {
1309  4         separatorString[i] = SEPARATOR_CHAR;
1310  3       }
1311  3       separatorString[count] = '\0';

1313  3       /*
1314  3        * Display the new state string and the separator
1315  3        */

1317  3       GTED_AppendStr ((TEdPtr)REST_ProgressWindow->NotifyMTed, "\n");
1318  3       GTED_AppendStr ((
                    TEdPtr)REST_ProgressWindow->NotifyMTed, stateString);
1319  3       GTED_AppendStr ((TEdPtr)REST_ProgressWindow->NotifyMTed, "\n");
1320  3       GTED_AppendStr ((
                  TEdPtr)REST_ProgressWindow->NotifyMTed, separatorString);
1321  3       GTED_AppendStr ((TEdPtr)REST_ProgressWindow->NotifyMTed, "\n");
1322  2     }

1324  2     notifyObject = EDMRST_GetFirstNotifyObject (
                                                  GREST_Handle, feedbackObject);
1325  2     while (notifyObject != NULL)
1326  3     {
1327  3       nextString = (Str) EDMRST_GetNofifyMessageText (GREST_Handle,
1328  3                                                       notifyObject);
1329  3       GTED_AppendStr ((
                    TEdPtr)REST_ProgressWindow->NotifyMTed, nextString);
1330  3       GTED_AppendStr ((TEdPtr)REST_ProgressWindow->NotifyMTed, "\n");
1331  3       notifyObject = EDMRST_GetNextNotifyObject (
                                                  GREST_Handle, notifyObject);
1332  2     }

1334  2     /* Flush all events */
1335  2     EVENT_ProcessPending ();

1337  2     if (!done)
1338  3     {
1339  3       if (!REST_ProgressDisplayed ||
1340  3           ((int)WGT_GetClientRes (
                              REST_ProgressWindow->ProgressArea) < 0))
1341  4       {
1342  4         EVENT_StartCBackAlarm (REST_CheckForCancel,
1343  4                                clientData,
1344  4                                RESTORE_CHECK_STARTUP_DELAY);
1345  3       }
1346  3       else
1347  4       {
1348  4         EVENT_StartCBackAlarm (REST_CheckForCancel,
1349  4                                clientData,
1350  4                                RESTORE_CHECK_CANCEL_DELAY);
1351  3       }

1353  3       if (currentState == RE_STATE_STOPPED)
1354  4       {
1355  4         REST_DisplayQuestion ();
1356  3       }
```

```
1357  2      }
1358  2      else
1359  3      {
1360  3        /* Display the Done dialog */
1361  3        REST_DisplayDone (status, feedbackObject);
1362  2      }

1364  2      /* Done with the feedback object */
1365  2      EDMRST_FreeFeedbackObject (GREST_Handle, &feedbackObject);
1366  1    }
1367       }
```



```
1369  /*********************************************************************
1370   * REST_GetSubmitResults
1371   *
1372   * Description:
1373   *   This routine get the results fo the submit call and starts
1374   *   the restore if successful.
1375   *
1376   * Parameters:
1377   *   clientData - The submit ID
1378   *
1379   * Returns:
1380   *   None.
1381   *
1382   *********************************************************************/

1384  static void REST_GetSubmitResults (ClientPtr  clientData)
1385  1 {
1386  1   unsigned int            submitId;
1387  1   eerrno_ty               status;
1388  1   unsigned long           objectsDone = 0;

1390  1   submitId = (unsigned int) clientData;

1392  1   /* Get the results of the submit */
1393  1   if ((status = EDMRST_GetSubmitResults (GREST_Handle,
1394  1                                          restoreCancelled,
1395  1                                          &submitId,
1396  1                                          &objectsDone)) ==
                                               EP_RB_RECOVER_RPC_INCOMPLETE)

1397  2   {
1398  2     if (!restoreCancelled)
1399  3     {
1400  3       /* Update progress */
1401  3       REST_ProgressUpdate (NULL);
1402  2     }

1404  2     /* Add a timeout to try again after a delay */
1405  2     EVENT_StartCBackAlarm (REST_GetSubmitResults,
1406  2                            (ClientPtr)submitId,
1407  2                            RESTORE_CHECK_SUBMIT_DELAY);
1408  1   }
1409  1   else
1410  2   {

1412  2     /* If submit did not succeed, display an error message */
1413  2     if (status != E_SUCCESS)
1414  3     {
1415  3       restoreInProgress = BOOL_FALSE;
1416  3       restoreCancelled = BOOL_FALSE;
1417  3       REST_SetRestoreVisibility (BOOL_TRUE);

1419  3       REST_ProgressDisplayError (status, NULL);
1420  2     }
1421  2     else if (!restoreCancelled)
1422  3     {
1423  3       /* Gentlemen start your engines! */
1424  3       if ((status = EDMRST_Start (
                                        GREST_Handle, submitId)) != E_SUCCESS)

1425  4       {
1426  4         /* Er uh, won't start, display the error message */
1427  4         restoreInProgress = BOOL_FALSE;
1428  4         REST_SetRestoreVisibility (BOOL_TRUE);
1429  4         REST_ProgressDisplayError (status, NULL);
1430  3       }
1431  3       else
```

```
1432  4         {
1433  4           /* Call the check for cancel routine to display the progress
                                                                      dialog */
1434  4           REST_CheckForCancel ((ClientPtr)&checkForCancelId);
1435  3         }
1436  2       }
1437  1     }
1438      }
```



```
1440    /*********************************************************************
1441     * REST_StartProgress
1442     *
1443     * Description:
1444     *   This routine will start the restore process and monitor its
                                                                      progres.
1445     *
1446     * Parameters:
1447     *   None.
1448     *
1449     * Returns:
1450     *   None.
1451     *
1452     *********************************************************************/

1454    void REST_StartProgress ( void )
1455  1 {
1456  1   REST_SetRestoreVisibility (BOOL_FALSE);

1458  1   REST_FirstDataTime = 0;

1460  1   REST_ProgressDisplay ();

1462  1   restoreInProgress = BOOL_TRUE;

1464  1   /* Add a timeout to get the submit results */
1465  1   EVENT_StartCBackAlarm (REST_GetSubmitResults,
1466  1                          (ClientPtr)0,
1467  1                          RESTORE_CHECK_SUBMIT_DELAY);
1468    }
```

```
/* -- Template created by NEURON DATA Open Interface.                    */
/* -- Do not alter 'CodeGen' directives.                                 */
/* (( CodeGen: GeneratorVersion 4 ))                                     */

/* -- Code generated on 06/30/99 at 10:43:40.                            */
/* -- Code regenerated on 07/22/99 at 13:35:35.                          */
/* -- Code regenerated on 07/22/99 at 13:36:30.                          */
/* -- Code regenerated on 08/12/99 at 14:39:17.                          */
/* -- Code regenerated on 08/13/99 at 10:57:20.                          */
/* -- Code regenerated on 08/16/99 at 10:34:32.                          */
/* -- Code regenerated on 08/30/99 at 15:12:28.                          */
/* -- Code regenerated on 08/31/99 at 07:58:51.                          */
/* -- Code regenerated on 08/31/99 at 08:02:13.                          */
/* -- Code regenerated on 08/31/99 at 08:10:32.                          */
/* -- Code regenerated on 11/10/99 at 15:59:13.                          */
/* (( CodeGen: CodeHistory ))                                            */

#define ERR_LIB RESTORE

#include "restProgress.h"
#include "gutil/guiUtils.h"
#include "gutil/tedUtils.h"
#include "gutil/winUtils.h"

ERR_EXTERN
ERR_INMODULE("RestProgress")

/* (( CodeGen: ClassImplementationPlaceHolder ))                         */
/* (( CodeGen: WinClassImplementationPlaceHolder ))                      */

/* (( CodeGen: WindowSection Win                                         */
/* ======================================================================= */
/* == Code for Window "Win"                                              */
/* ======================================================================= */

/* (( CodeGen: MenuImplementationPlaceHolder ))                          */

/* (( CodeGen: WgtNfyHandler ValidateWiNameTEd                           */
static   void C_FAR RestProgressWin_ValidateWiNameTEd L1(
                                                RestProgressWinPtr, win)
{
}
/* )) CodeGen: WgtNfyHandler ValidateWiNameTEd
```



```
                                                                      */

/* (( CodeGen: WgtNfyHandler HitCancelTBut                               */
static   void C_FAR RestProgressWin_HitCancelTBut L1(
                                                RestProgressWinPtr, win)
{
  REST_ProgressCancel (BOOL_TRUE);
}
/* )) CodeGen: WgtNfyHandler HitCancelTBut                               */

/* (( CodeGen: WgtNfyHandler ValidateStartTimeTEd                        */
static   void C_FAR RestProgressWin_ValidateStartTimeTEd L1(
                                                RestProgressWinPtr, win)
{
}
/* )) CodeGen: WgtNfyHandler ValidateStartTimeTEd                        */

/* (( CodeGen: WgtNfyHandler ValidateEndTimeTEd                          */
static   void C_FAR RestProgressWin_ValidateEndTimeTEd L1(
                                                RestProgressWinPtr, win)
{
}
/* )) CodeGen: WgtNfyHandler ValidateEndTimeTEd                          */

/* (( CodeGen: WgtNfyHandler ValidateStatusTed                           */
static   void C_FAR RestProgressWin_ValidateStatusTed L1(
                                                RestProgressWinPtr, win)
{
}
/* )) CodeGen: WgtNfyHandler ValidateStatusTed                           */

/* (( CodeGen: WgtNfyHandler ValidateNotifyMTed                          */
static   void C_FAR RestProgressWin_ValidateNotifyMTed L1(
                                                RestProgressWinPtr, win)
{
}
/* )) CodeGen: WgtNfyHandler ValidateNotifyMTed                          */
/* (( CodeGen: WgtNfyPlaceHolder ))                                      */

/* (( CodeGen: UseDefaultNfyHandler name_of_nfy_handler ))               */
/* (( CodeGen: UseAllDefaultNfyHandlers name_of_wgt_member ))            */

static  void RestProgressWin_RedrawProgressArea (
                                                RestProgressWinPtr win)
{
  WGT_DefNfy ((WgtPtr)win->ProgressArea, WGT_NFYREDRAW);
  REST_ProgressDrawPercentComplete ((WgtPtr)win->ProgressArea);
}

void     RestProgressWin_Construct L1(RestProgressWinPtr, win)
{
         /* ((
         CodeGen: WgtInitializations                                  */
```

```
88  1           win->ProgressPanel = (PanelPtr)PANEL_GetNamedWgt((
                                                 PanelPtr)win, "ProgressPanel");
89  1           win->WiNameTArea = (TAreaPtr)PANEL_GetNamedWgt((
                                                  PanelPtr)win, "WiNameTArea");
90  1           win->WiNameTEd = (STEdPtr)PANEL_GetNamedWgt((
                                                   PanelPtr)win, "WiNameTEd");
91  1           win->ProgressArea = (PanelPtr)PANEL_GetNamedWgt((
                                                 PanelPtr)win, "ProgressArea");
92  1           win->StartTimeTEd = (STEdPtr)PANEL_GetNamedWgt((
                                                 PanelPtr)win, "StartTimeTEd");
93  1           win->EndTimeTEd = (STEdPtr)PANEL_GetNamedWgt((
                                                  PanelPtr)win, "EndTimeTEd");
94  1           win->StatusTed = (MTEdPtr)PANEL_GetNamedWgt((
                                                  PanelPtr)win, "StatusTed");
95  1           win->NotifyMTed = (MTEdPtr)PANEL_GetNamedWgt((
                                                  PanelPtr)win, "NotifyMTed");
96  1           win->CancelTBut = (PButPtr)PANEL_GetNamedWgt((
                                                  PanelPtr)win, "CancelTBut");
97  1           WIN_SetWgtNfyHandler((WinPtr)win, (
                                  WgtPtr)win->WiNameTEd, TED_NFYVALIDATE,
98  1                   (
                     WinWgtNfyHandlerProc)RestProgressWin_ValidateWiNameTEd);
99  1           WIN_SetWgtNfyHandler((WinPtr)win, (
                                 WgtPtr)win->StartTimeTEd, TED_NFYVALIDATE,
100 1                   (
                   WinWgtNfyHandlerProc)RestProgressWin_ValidateStartTimeTEd);
101 1           WIN_SetWgtNfyHandler((WinPtr)win, (
                                  WgtPtr)win->EndTimeTEd, TED_NFYVALIDATE,
102 1                   (
                    WinWgtNfyHandlerProc)RestProgressWin_ValidateEndTimeTEd);
103 1           WIN_SetWgtNfyHandler((WinPtr)win, (
                                  WgtPtr)win->StatusTed, TED_NFYVALIDATE,
104 1                   (
                     WinWgtNfyHandlerProc)RestProgressWin_ValidateStatusTed);
105 1           WIN_SetWgtNfyHandler((WinPtr)win, (
                                 WgtPtr)win->NotifyMTed, TED_NFYVALIDATE,
106 1                   (
                    WinWgtNfyHandlerProc)RestProgressWin_ValidateNotifyMTed);
107 1           WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->CancelTBut, TBUT_NFYHIT,
108 1                   (WinWgtNfyHandlerProc)RestProgressWin_HitCancelTBut);
109 1           /* )) CodeGen: WgtInitializations 128970
                                                                          */

111 1     WGT_SetNfyProc ((WgtPtr)win->WiNameTEd,    GUTIL_OutputTED_DefNfy);
112 1     WGT_SetNfyProc ((WgtPtr)win->StartTimeTEd, GUTIL_OutputTED_DefNfy);
113 1     WGT_SetNfyProc ((WgtPtr)win->EndTimeTEd,   GUTIL_OutputTED_DefNfy);
114 1     WGT_SetNfyProc ((WgtPtr)win->StatusTed,    GUTIL_OutputTED_DefNfy);
115 1     WGT_SetNfyProc ((WgtPtr)win->NotifyMTed,   GUTIL_OutputTED_DefNfy);

117 1     WIN_SetWgtNfyHandler((WinPtr)win, (WgtPtr)win, WGT_NFYRESIZE,
118 1                          (WinWgtNfyHandlerProc)GUTIL_WinHandleResize);
119 1     WIN_SetWgtNfyHandler((WinPtr)win, (WgtPtr)win, WIN_NFYMOUSECLICK,
120 1                          (
                            WinWgtNfyHandlerProc)GUTIL_WinHandleMouseClick);

122 1           WIN_SetWgtNfyHandler((WinPtr)win,
123 1                           (WgtPtr)win->ProgressArea,
124 1                           PANEL_NFYREDRAW,
125 1                          (
                    WinWgtNfyHandlerProc)RestProgressWin_RedrawProgressArea);


127       }

129       RestProgressWinPtr       REST_ProgressWindowInit (WinPtr parent)
130 1     {
```



```
131 1           RestProgressWinPtr win;

133 1           win = (RestProgressWinPtr)WIN_LoadSized(
                                                    "RestProgress", "Win",
134 1                   sizeof(RestProgressWinRec));
135 1           RestProgressWin_Construct(win);

137 1     /* Set this module to be non-visible when closed */
138 1     if (parent != NULL)
139 2     {
140 2       WIN_SetParentWin ((WinPtr)win, parent);
141 2       WIN_SetOptFlags  ((WinPtr)win, WIN_OPTABOVEPARENT);
142 2       WIN_SetIconFlags ((WinPtr)win, WIN_OPTABOVEPARENT);
143 1     }

145 1           WIN_Init((WinPtr)win);

147 1     return (win);
148       }

150       /* )) CodeGen: WindowSection Win                                  */

151       /* (( CodeGen: WindowImplementationPlaceHolder ))
                                                                          */

153       /* (( CodeGen: MainPlaceHolder ))
                                                                          */
```

```
/*****************************************************************************
 * restSearchMgr.c
 *
 *
 * Copyright 1996 by Epoch Systems, Inc.
 *
 *
 * Mission Statement:
 *   This file contains the functions necessary for the EDM Restore
 *   Search window.
 *            Note that searching only works for file system workitems,
 *   no OLDB support is included.
 *
 * Required includes:
 *   None
 *
 * Compile-Time Options:
 *   N/A
 *
 *
 * RCS Information:
 *     $RCSfile$
 *     $Revision$
 *     $Date$
 *****************************************************************************/

#define  ERR_LIB          RESTORE

#include <esl/c_portable.h>
#include <esl/ep_xopen.h>

#include <respub.h>
#include <cboxpub.h>
#include <panelpub.h>
#include <tbutpub.h>
#include <winpub.h>
#include <drawpub.h>

#include "eerrno/e_errno.h"
#include "util/esl_string.h"
#include <restore/restore_api.h>

#include "restore.h"
#include "restoreP.h"
#include "restUtils.h"
#include "restSelMgr.h"
#include "restCalendar.h"
#include "restSearch.h"
#include "restAPIUtils.h"
#include "gutil/timeUtils.h"
#include "gutil/alertMgr.h"
#include "gutil/iconDefs.h"
#include "gutil/iconUtils.h"
#include "gutil/winUtils.h"
#include "gutil/lboxUtils.h"
#include "gutil/guiUtils.h"

/*************
 * Constants *
 *************/

#define MAX_FOUND_OBJECTS     10    /* Maximum objects to find per
                                                          iteration */
#define NUMBER_FOUND_COLUMNS 10    /* Number of columns in the list box
                                                                    */
```



```
#define BACKUP_TIME_COLUMN 1
#define MARK_COLUMN        2
#define ICON_COLUMN        3
#define NAME_COLUMN        4
#define PERMISSIONS_COLUMN 5
#define OWNER_COLUMN       6
#define GROUP_COLUMN       7
#define SIZE_COLUMN        8
#define DATE_COLUMN        9
#define TIME_COLUMN       10

/*
 * Number of milliseconds to delay
 */

#define SEARCH_ALARM_DELAY                100
#define RESTORE_START_FIND_RESULTS_DELAY 500
#define RESTORE_CONT_FIND_RESULTS_DELAY   10

/*************************
 * Local Data structures *
 *************************/

typedef struct _REST_FoundObjectRec
{
  time_t       backupTime;
                                /* The time of the backup for this object */
  GREST_Object object;              /* The object */
  struct _REST_FoundObjectRec *next;/* The next found object */
} REST_FoundObjectRec, *REST_FoundObjectPtr;

/********************
 * Global Variables *
 ********************/

/* Icons for objects */
static IconPtr searchDirIcon;
static IconPtr searchFileIcon;
static IconPtr searchCheckBoxIcon;
static IconPtr searchCheckIcon;
static IconPtr searchSelCheckIcon;
static IconPtr searchOffsiteIcon;
static IconPtr searchBadIcon;

/* First and last objects in the list */
static REST_FoundObjectPtr firstFoundObject = NULL;
static REST_FoundObjectPtr lastFoundObject = NULL;

/* Number of found entries */
static unsigned long REST_CurrentEntries = 0;

/* Flags for current status */
static BoolEnum REST_SearchTerminated = BOOL_FALSE;
static BoolEnum REST_SearchDisplayed = BOOL_FALSE;

/* Handle to the current search in progress dialog */
static GALERT_WinHandle synchSearchHandle = NULL;

/* Flags for search in progress */
static BoolEnum searchInProgress = BOOL_FALSE;

static long REST_SearchCookie = INIT_COOKIE;
                                                  /* Ah, the magic cookie */

/*****************************************************************************
```

```
128   * Function Name:    REST_AddFoundItem ()
129   *
130   * Description:
131   *    This routine will add a found object to the search results
132   *    list box (at the end of the list box).
133   *
134   * Parameters:
135   *    object      (I) - The object to add
136   *    backupTime (I) - The backup time for the object
137   *
138   * Success Outputs and Side Effects:
139   *    None.
140   *
141   * Returns:
142   *    None.
143   *
144   ************************************************************************/

146   static void REST_AddFoundItem (GREST_Object object,
147                                  time_t        backupTime)
148 1 {
149 1   REST_FoundObjectPtr foundObject;  /* New found object record */

151 1   /* Create a new found object record */
152 1   foundObject = (REST_FoundObjectPtr) GUTIL_Malloc(sizeof(
                                                 REST_FoundObjectRec));

154 1   /* Fill in the fields */
155 1   foundObject->backupTime = backupTime;
156 1   foundObject->object = object;
157 1   foundObject->next = NULL;

159 1   /* Put this on the end of the list of found objects */
160 1   if (lastFoundObject != NULL)
161 2   {
162 2     /* Tack it onto the end */
163 2     lastFoundObject->next = foundObject;
164 2     lastFoundObject = foundObject;
165 1   }
166 1   else
167 2   {
168 2     /* This is the only object, first and last */
169 2     firstFoundObject = foundObject;
170 2     lastFoundObject = foundObject;
171 1   }
172   }
```



```
174   /************************************************************************
175    * REST_CheckForCancel
176    *
177    * Description:
178    *    This routine will determine if the user has cancelled the search
                                                                        and
179    *    will update the progress window.
180    *
181    * Parameters:
182    *    None.
183    *
184    * Returns:
185    *    BOOL_TRUE  - If the user has cancelled
186    *    BOOL_FALSE - otherwise
187    *
188    ************************************************************************/

190   static BoolEnum REST_CheckForCancel (void)
191 1 {
192 1   static long lastNumFound = -1;
                                        /* Number of items found previously */
193 1   Char        outputString[MAX_STRING_LENGTH];
                                       /* Updated string to display */
194 1   BoolEnum    retVal;                      /* Value to return */

196 1   /* Make sure the cancel dialog is still displayed */
197 1   if (synchSearchHandle != NULL)
198 2   {
199 2     /* If the number of entries found has changed,
                                                    update the string */
200 2     if (REST_CurrentEntries != lastNumFound)
201 3     {
202 3       /* Build the new string */
203 3       if (REST_CurrentEntries != 0)
204 4       {
205 4         STR_Sprintf (outputString,
206 4                      REST_GetErrorString (REST_SEARCH_STATUS),
207 4                      REST_CurrentEntries);
208 3       }
209 3       else
210 4       {
211 4         STR_Cpy (outputString, REST_GetErrorString (
                                              REST_SEARCH_IN_PROGRESS));
212 3       }

214 3       /* Update the progress dialog */
215 3       GALERT_UpdateMessage (synchSearchHandle, outputString);

217 3       /* Save the current number of entries */
218 3       lastNumFound = REST_CurrentEntries;
219 2     }

221 2     /* Return whether or not the user cancelled */
222 2     retVal = GALERT_IsCancelled(synchSearchHandle);
223 1   }
224 1   else
225 2   {
226 2     retVal = BOOL_TRUE;
227 1   }

229 1   return (retVal);
230   }
```

```
232  /*****************************************************************
233   * REST_SearchStartTimeout
234   *
235   * Description:
236   *   This routine will start the search.
                                        It is used to delay a bit to make
237   *   sure the in progress dialog is visible.
238   *
239   * Parameters:
240   *   clientData (I) - Unused.
241   *
242   * Returns:
243   *   None.
244   *
245   *****************************************************************/

247  static void REST_SearchStartTimeout (ClientPtr clientData)
248 1 {
249 1   REST_SearchStartSearch ();
250  }
```



```
252  /*****************************************************************
253   * REST_SearchCancel
254   *
255   * Description:
256   *   This routine will remove the search in progress dialog and clear
257   *   the current sync window handle.
258   *
259   * Parameters:
260   *   None.
261   *
262   * Returns:
263   *   None.
264   *
265   *****************************************************************/

267  void REST_SearchCancel (void)
268 1 {
269 1   long          numEntries;         /* Number found */
270 1   GREST_Object  foundObjects[1];    /* Temp array of found objects */
271 1   time_t        times[1];           /* Backup time for object */
272 1   eerrno_ty     eerrno;             /* Error code returned */

274 1   /* If there really is a search in progress, remove the dialog */
275 1   if (synchSearchHandle != NULL)
276 2   {
277 2     GALERT_CancelSynchDialog (synchSearchHandle);
278 2     synchSearchHandle = NULL;
279 1   }

281 1   /*
282 1    * If there is a search in progress then cancel the find operation
283 1    */

285 1   if (searchInProgress)
286 2   {
287 2     /* Cancel the find operation, ignore results */
288 2     EDMRST_GetFindResults (GREST_Handle,
289 2                            BOOL_TRUE,
290 2                            1,
291 2                            foundObjects,
292 2                            times,
293 2                            &numEntries,
294 2                            &REST_SearchCookie);
295 1   }
296  }
```

```
298  /************************************************************************
299   * REST_DisplaySearchInProgress
300   *
301   * Description:
302   *   This routine will display the search in progress dialog and
303   *   initialize the current number of entries found.
304   *
305   * Parameters:
306   *   None.
307   *
308   * Returns:
309   *   None.
310   *
311   ************************************************************************/

313  void REST_DisplaySearchInProgress (void)
314 1{
315 1   time_t  startTime;  /* Current start time */
316 1   time_t  endTime;    /* Current end time */

318 1   /* Do nothing if we're already searching,
                                        or a restore is in progress */
319 1   if (REST_SearchInProgress() || REST_RestoreInProgress())
320 1     return;

322 1   /* Get the start backup date/time */
323 1   startTime = (time_t)WGT_GetClientRes ((
                             WgtPtr)REST_SearchWindow->StartDateOutput);

325 1   /* Get the end backup date/time */
326 1   endTime = (time_t)WGT_GetClientRes ((
                              WgtPtr)REST_SearchWindow->EndDateOutput);

328 1   /* Validate the search start and end times */
329 1   if (startTime > endTime)
330 2   {
331 2     /* The start time is after the end time,
                                           no searching can be done */
332 2     GALERT_DisplayError ((WinPtr)REST_SearchWindow,
333 2                          REST_GetErrorString (REST_SEARCH_FAIL_TITLE),
334 2                          GICON_GetError(),
335 2                          REST_GetErrorString (REST_SEARCH_TIME_ERROR));
336 1   }
337 1   else
338 2   {

340 2     /* Initialize the number of current entries to zero */
341 2     REST_CurrentEntries = 0;

343 2     /* Initialize the window */
344 2     synchSearchHandle = GALERT_DisplaySynchronousWait
345 2                           ((WinPtr)REST_SearchWindow,
346 2                            REST_GetErrorString (
                                        REST_SEARCH_PROGRESS_TITLE),
347 2                            GICON_GetIcon (I_WAIT),
348 2                            REST_GetErrorString (
                                           REST_SEARCH_IN_PROGRESS),
349 2                            BOOL_TRUE);

351 2     /* Delay a bit to make sure the dialog is visible */
352 2     EVENT_StartCBackAlarm (REST_SearchStartTimeout,
353 2                            (ClientPtr)NULL,
354 2                            SEARCH_ALARM_DELAY);
355 1   }
```



```
357  }
```

```
359  /*****************************************************************
360   * REST_SearchGetFoundItem
361   *
362   * Description:
363   *   This routine will return the found item that would appear in the
364   *   given row.
365   *
366   * Parameters:
367   *   row (I) - The row to find the item in
368   *
369   * Returns:
370   *   REST_FoundObjectPtr - Item in row if row is valid
371   *                       - NULL if invalid row,
                                          <= 0 or > number of rows
372   *
373   *****************************************************************/

375  static REST_FoundObjectPtr REST_SearchGetFoundItem (Int32 row)
376 1 {
377 1   REST_FoundObjectPtr foundObject = NULL;
378 1   REST_FoundObjectPtr nextObject = NULL;
379 1   Int32               count = 1;

381 1   /* Validate the row first */
382 1   if (row > 0)
383 2   {
384 2     /* get to the correct object in the list */
385 2     nextObject = firstFoundObject;
386 2     while ((nextObject != NULL) && (count < row))
387 3     {
388 3       nextObject = nextObject->next;
389 3       count ++;
390 2     }

392 2     foundObject = nextObject;
393 1   }

395 1   /* Return the appropriate object */
396 1   return (foundObject);
397  }
```



```
399  /*****************************************************************
400   * REST_SearchUpdateButtons
401   *
402   * Description:
403   *   This routine will update the sensitivity of the search results
404   *   action buttons (
                        mark, unmark, set view, clear) based on the current
405   *   selections.
406   *
407   * Parameters:
408   *   None
409   *
410   * Returns:
411   *   None.
412   *
413   *****************************************************************/

415  void REST_SearchUpdateButtons (void)
416 1 {
417 1   Int32               foundRow;
                                          /* Current row selection */
418 1   REST_FoundObjectPtr foundObject;
                                          /* Next object in the list */
419 1   BoolEnum            itemsExist = BOOL_FALSE;
                                          /* Flag if there are items */
420 1   int                 selectedCount = 0;
                                          /* Number of selected items */
421 1   BoolEnum            unmarkEnabled = BOOL_FALSE;
                                          /* Flag to unable unmark */
422 1   BoolEnum            markEnabled = BOOL_FALSE;
                                          /* Flag to unable mark */
423 1   BoolEnum            setViewEnabled = BOOL_FALSE;
                                          /* Flag to unable set view */

425 1   /* If a restore is in progress, don't update any sensitivity */
426 1   if (REST_RestoreInProgress())
427 1     return;

429 1   /*
430 1    * Loop through the list examining the selected items
431 1    */

433 1   /* Get the object at the first selected row position */
434 1   foundRow = GUTIL_LBOX_GetFirstVLboxSelectedRow (
                                          REST_SearchWindow->FoundBox);
435 1   foundObject = REST_SearchGetFoundItem (foundRow);

437 1   while (foundObject != NULL)
438 2   {
439 2     /* At least we know there are entries in the list */
440 2     itemsExist = BOOL_TRUE;

442 2     /* Bump the selected count */
443 2     selectedCount++;

445 2     /* If the object is not marked, determine if it is markable */
446 2     if (!GREST_IsObjectMarkedForTime (GREST_Handle,
447 2                                       foundObject->object,
448 2                                       foundObject->backupTime))
449 3     {

451 3       /* Determine if this object is markable */
452 3       if (GREST_IsObjectMarkableForTime (GREST_Handle,
453 3                                          foundObject->object,
454 3                                          foundObject->backupTime) &&
```

```
455 3           ((EDMRST_GetObjectStatus (GREST_Handle,
456 3                                     foundObject->object) !=
                                                        Backup_Bad) ||
457 3            REST_MarkBadFiles))
458 4         {
459 4           /* It is markable, so enable the mark button */
460 4           markEnabled = BOOL_TRUE;
461 3         }
462 2       }

464 2       /* Else this object is marked so enable the unmark button */
465 2       else
466 3       {
467 3         unmarkEnabled = BOOL_TRUE;
468 2       }

470 2       /* Get the object at the next selected row position */
471 2       foundRow = GUTIL_LBOX_GetNextVLboxSelectedRow (
                                              REST_SearchWindow->FoundBox);
472 2       foundObject = REST_SearchGetFoundItem (foundRow);
473 1     }

475 1     /* If any items were in the list */
476 1     if (itemsExist)
477 2     {
478 2       /* Only enable the set view button if only one object is selected
                                                                      */
479 2       if (selectedCount == 1)
480 2         setViewEnabled = BOOL_TRUE;
481 1     }

483 1     /* Apply what we have determined */
484 1     WGT_SetEnabled ((WgtPtr)REST_SearchWindow->MarkButton, markEnabled);
485 1     WGT_SetEnabled ((
                  WgtPtr)REST_SearchWindow->UnMarkButton, unmarkEnabled);
486 1     WGT_SetEnabled ((
                 WgtPtr)REST_SearchWindow->SetViewButton, setViewEnabled);
487   }
```



```
489   /*********************************************************************
490    * REST_SearchFillLBox
491    *
492    * Description:
493    *   This routine will fill the virtual listbox rows given the start
494    *   row (number in row 1) and the bounds of the list box rows.
495    *
496    * Parameters:
497    *   lbox     (I) - The virtual list box to fill
498    *   startRow (I) - The real row number that is now in row 1
499    *   bounds   (I) - The number of rows in the list box
500    *
501    * Returns:
502    *   None.
503    *
504    *********************************************************************/

506   static void REST_SearchFillLBox (LBoxPtr  lbox,
507                                    Int32    startRow,
508                                    Int16    bounds)
509 1 {
510 1   REST_FoundObjectPtr nextObject;
511 1   ClientPtr           clientData;
512 1   Int32               count = 1;
513 1   Int16               row;

515 1   /* get to the correct object in the list */
516 1   nextObject = firstFoundObject;
517 1   while ((nextObject != NULL) && (count < startRow))
518 2   {
519 2     nextObject = nextObject->next;
520 2     count ++;
521 1   }

523 1   /* Put the next set of rows in the lbox, even if NULL */
524 1   REST_SearchLboxFrozen = BOOL_TRUE;
525 1   row = 1;
526 1   while (row <= bounds)
527 2   {
528 2     /* Go to the row (rows start at 1) */
529 2     LBOX_GoCol1Row (lbox, row);

531 2     /* Set the client data for this row */
532 2     LBOX_CurSetClientData (lbox, (ClientPtr)nextObject);

534 2     /* go to the next object */
535 2     if (nextObject != NULL)
536 2       nextObject = nextObject->next;
537 2     row++;
538 1   }
539 1   REST_SearchLboxFrozen = BOOL_FALSE;


542 1   /* Now fit in the client data */
543 1   LBOX_GoHome (REST_SearchWindow->FoundBox);
544 1   row = 1;
545 1   while ((clientData = LBOX_CurGetClientData (
                                    REST_SearchWindow->FoundBox)) != NULL)
546 2   {
547 2     GUTIL_FitThisDataInRow (REST_SearchWindow->FoundBox,
548 2                             clientData,
549 2                             row,
550 2                             NUMBER_FOUND_COLUMNS,
551 2                             REST_GetSearchListColumnValues);
552 2     row++;
```

```
553 2     LBOX_GoCol1Row (REST_SearchWindow->FoundBox, row);
554 1   }
555   }
```



```
557   /*****************************************************************
558    * REST_SearchDisplay
559    *
560    * Description:
561    *   This routine will display the search window, with the starting
562    *   search directory to the given directory (/ if NULL).
563    *
564    * Parameters:
565    *   currentDirectory (I) - Directory to start the search from
566    *
567    * Returns:
568    *   None.
569    *
570    *****************************************************************/

572   void REST_SearchDisplay (Str currentDirectory)
573 1 {
574 1   time_t         currentTime;
                                                    /* Current backup time */
575 1   Char           dateString[SMALL_STRING_LENGTH];
                                                    /* String for the date */
576 1   GREST_Object   currentWI;
                                                    /* Work-item to search */
577 1   Char           windowLabel[MAX_STRING_LENGTH];
                                                   /* Lable for the window */
578 1   Char           name[MAX_CLIENT_NAME_LENGTH];
                                                  /* Name of the Work-Item */

580 1   if (!REST_SearchDisplayed)
581 2   {
582 2     /* Load up the window's resources */
583 2     REST_SearchWindowInit ();

585 2     /* Set buttons and labels to default values */
586 2     TED_SetStr ((
              TEdPtr)REST_SearchWindow->DirectoryText, currentDirectory);
587 2     TBUT_SetSelected ((
                    TButPtr)REST_SearchWindow->DescendToggle, BOOL_TRUE);
588 2     TED_ClearAll ((TEdPtr)REST_SearchWindow->StringText);
589 2     TBUT_SetSelected ((
                     TButPtr)REST_SearchWindow->IncludeRadio, BOOL_TRUE);
590 2     TBUT_SetSelected ((
                    TButPtr)REST_SearchWindow->ExcludeRadio, BOOL_FALSE);
591 2     TED_ClearAll ((TEdPtr)REST_SearchWindow->OwnerText);
592 2     TBUT_SetSelected ((
                TButPtr)REST_SearchWindow->IncludeOwnerToggle, BOOL_TRUE);
593 2     TBUT_SetSelected ((
               TButPtr)REST_SearchWindow->ExcludeOwnerToggle, BOOL_FALSE);
594 2     TED_ClearAll ((TEdPtr)REST_SearchWindow->GroupText);
595 2     TBUT_SetSelected ((
                TButPtr)REST_SearchWindow->IncludeGroupToggle, BOOL_TRUE);
596 2     TBUT_SetSelected ((
              TButPtr)REST_SearchWindow->ExcludeGroupToggle, BOOL_FALSE);
597 2     TBUT_SetSelected ((
                  TButPtr)REST_SearchWindow->AllTypesToggle, BOOL_TRUE);
598 2     TBUT_SetSelected ((
                 TButPtr)REST_SearchWindow->AllStatusToggle, BOOL_TRUE);
599 2     TED_ClearAll ((TEdPtr)REST_SearchWindow->SizeText);
600 2     TBUT_SetSelected ((
                TButPtr)REST_SearchWindow->GreaterThanToggle, BOOL_FALSE);
601 2     TBUT_SetSelected ((
                 TButPtr)REST_SearchWindow->LessThanToggle, BOOL_FALSE);
602 2     TBUT_SetSelected ((
                    TButPtr)REST_SearchWindow->EqualsToggle, BOOL_TRUE);
```

```
604  2      /* Set the date strings to the current backup date */
605  2      if (EDMRST_GetCurrentBackupTime (GREST_Handle, &currentTime) == 0)
606  3      {
607  3        strftime (dateString,
608  3                  SMALL_STRING_LENGTH,
609  3                  "%b %d %H:%M",
610  3                  localtime (&currentTime));
611  2      }
612  2      else
613  3      {
614  3        /* No current time (?), blank out the string */
615  3        STR_Cpy (dateString, "");
616  2      }

618  2      TED_SetStr ((
                    TEdPtr)REST_SearchWindow->StartDateOutput, dateString);
619  2      TED_SetStr ((TEdPtr)REST_SearchWindow->EndDateOutput, dateString);
620  2      WGT_SetClientRes ((WgtPtr)REST_SearchWindow->StartDateOutput, (
                                                    ClientPtr)currentTime);
621  2      WGT_SetClientRes ((WgtPtr)REST_SearchWindow->EndDateOutput, (
                                                    ClientPtr)currentTime);


623  2      /* Make the list box a virtual list box */
624  2      GUTIL_LBOX_SetVirtual (REST_SearchWindow->FoundBox,
625  2                             REST_SearchWindow->FoundSbar,
626  2                             24,
627  2                             NULL,
628  2                             REST_SearchFillLBox);

630  2      /* We ain't found nothing yet */
631  2      REST_ClearFoundBox ();

633  2      /* Set the status flag */
634  2      REST_SearchTerminated = BOOL_FALSE;

636  2      /* Set the label for the window */
637  2      currentWI = REST_GetCurrentWorkItem ();
638  2      STR_Cpy (name, EDMRST_GetObjectFullName (GREST_Handle, currentWI));
639  2      GUTIL_SetDefaultWindowTitle ((WinPtr)REST_SearchWindow, name);

641  2      REST_SearchDisplayed = BOOL_TRUE;
642  1    }

644  1    /* Put up the window */
645  1    WIN_Show ((WinPtr)REST_SearchWindow);

647    }
```



```
649    /*********************************************************************
650     * Function Name:   REST_SearchDisposeInfo ()
651     *
652     * Description:
653     *   This routine will free the info in the current cell.
654     *
655     * Parameters:
656     *   info (I) - The found object info pointer.
657     *
658     * Success Outputs and Side Effects:
659     *   None.
660     *
661     * Returns:
662     *   None.
663     *
664     *********************************************************************/

666    static void REST_SearchDisposeInfo (REST_FoundObjectPtr info)
667  1 {

669  1   /* Get the info */
670  1   if (info == NULL)
671  2   {
672  2     /* Free the restore object */
673  2     EDMRST_FreeRestorableObjects (GREST_Handle, &info->object, 1);

675  2     /* Free the info */
676  2     GUTIL_Free (info);
677  1   }
678    }
```

```
680   /*********************************************************************
681    * Function Name:   REST_SearchRemove ()
682    *
683    * Description:
684    *   This routine will remove the search window.
685    *
686    * Parameters:
687    *   None.
688    *
689    * Success Outputs and Side Effects:
690    *   None.
691    *
692    * Returns:
693    *   None.
694    *
695    *********************************************************************/

697   BoolEnum REST_SearchRemove (void)
698 1 {
699 1   BoolEnum            returnStatus;  /* Duh, status to remove */
700 1   REST_FoundObjectPtr thisObject;
                                /* Pointer to found object to dispose */
701 1   REST_FoundObjectPtr nextObject;
                                      /* Next found object in the list */

703 1   /* Do nothing if a restore is in progress */
704 1   if (REST_RestoreInProgress())
705 2   {
706 2     return (BOOL_FALSE);
707 1   }

709 1   /* We only care if the window is displayed */
710 1   if (REST_SearchDisplayed)
711 2   {
712 2     /* If the user is currently searching, stop the search */
713 2     REST_SearchCancel ();

715 2     /* Stop any alarms that were set */
716 2     EVENT_StopCBackAlarm ((ClientPtr)GREST_Handle);

718 2     REST_SearchDisplayed = BOOL_FALSE;

720 2     /* Remove the window, but no need to tell us about it */
721 2     WIN_SelfTerminate ((WinPtr)REST_SearchWindow);

723 2     /* Free the current found list */
724 2     thisObject = firstFoundObject;
725 2     while (thisObject != NULL)
726 3     {
727 3       nextObject = thisObject->next;
728 3       REST_SearchDisposeInfo (thisObject);
729 3       thisObject = nextObject;
730 2     }
731 2     firstFoundObject = NULL;
732 2     lastFoundObject = NULL;

734 2     REST_SearchWindow = NULL;

736 2     returnStatus = BOOL_TRUE;
737 1   }
738 1   else
739 2   {
740 2     returnStatus = BOOL_FALSE;
741 1   }
```



```
743 1   /* Return whether or now we actually had to remove the window */
744 1   return (returnStatus);
745   }
```

```
747  /*****************************************************************
748   * Function Name:   REST_SearchWindowIsDisplayed ()
749   *
750   * Description:
751   *   This routine will determin if the search window is currently
752   *   displayed.
753   *
754   * Parameters:
755   *   None.
756   *
757   * Success Outputs and Side Effects:
758   *   None.
759   *
760   * Returns:
761   *   BOOL_TRUE  - If the window is displayed
762   *   BOOL_FALSE - otherwise.
763   *
764   *****************************************************************/

766  BoolEnum REST_SearchWindowIsDisplayed (void)
767 1 {
768 1   return (REST_SearchDisplayed);
769   }
```



```
771  /*****************************************************************
772   * Function Name:   REST_SearchGetTime ()
773   *
774   * Description:
775   *   This routine will query the user for a backup date/time and
                                                                update
776   *   the date edit widget with the new date if selected.
777   *
778   * Parameters:
779   *   timeTEd (I) - The widget to set the time in.
780   *
781   * Success Outputs and Side Effects:
782   *   None.
783   *
784   * Returns:
785   *   None.
786   *
787   *****************************************************************/

789  void REST_SearchGetTime (TEdPtr  timeTEd)
790 1 {
791 1   Char          dateString[SMALL_STRING_LENGTH];  /* New date string */
792 1   GREST_Object currentWI;
                                                    /* Current work item */
793 1   time_t        currentTime;
                                                  /* Current backup time */
794 1   time_t        newTime = 0;                      /* New backup time */

796 1   /* Do nothing if we're searching, or a restore is in progress */
797 1   if (REST_SearchInProgress() || REST_RestoreInProgress())
798 1     return;

800 1   /* Get the currently selected time */
801 1   currentTime = (time_t)WGT_GetClientRes ((WgtPtr)timeTEd);

803 1   /* Get the work item */
804 1   currentWI = REST_GetCurrentWorkItem ();
805 1   if (currentWI != NULL)
806 2   {

808 2     /* Query the user for the new start time */
809 2     newTime = REST_GetUserSelectedTime (currentWI,
810 2                                         currentTime,
811 2                                         (WinPtr)REST_SearchWindow);
812 1   }

814 1   /* If the user selected a time */
815 1   if (newTime != 0)
816 2   {
817 2     /* Get the new date string */
818 2     strftime (dateString,
819 2               SMALL_STRING_LENGTH,
820 2               "%b %d %H:%M",
821 2               localtime (&newTime));

823 2     TED_SetStr ((TEdPtr)timeTEd, dateString);
824 2     WGT_SetClientRes ((WgtPtr)timeTEd, (ClientPtr)newTime);
825 1   }
826  }
```

```
828  /*****************************************************************
829   * REST_GetSearchListColumnValues
830   *
831   * Description:
832   *   This routine will return the drawables for the given information
833   *   for the given search results list box column.
834   *
835   * Parameters:
836   *   lbox          (I) - The list box to draw to (Not Used)
837   *   clientData    (I) - The data in the cell. (
                                              a private data structure)
838   *   row           (I) - The row of the cell
839   *   column        (I) - The column of the cell
840   *   text          (O) - The text to draw in the cell.
841   *   justification (O) - The justification of the text to be drawn.
842   *   icon1         (O) - The first icon to draw in the cell.
843   *   icon2         (O) - The second icon to draw in the cell.
844   *   overlayIcon1  (O) - The first overlayed icon to draw in the cell.
845   *   overlayIcon2  (
                         O) - The second overlayed icon to draw in the cell.
846   *
847   * Returns:
848   *   BOOL_TRUE  - if the cell should be drawn
849   *   BOOL_FALSE - otherwise
850   *
851   *****************************************************************/

853  BoolEnum REST_GetSearchListColumnValues (LBoxPtr   lbox,
854                                           ClientPtr clientData,
855                                           Int16     row,
856                                           Int16     column,
857                                           Str       text,
858                                           UInt16    *justification,
859                                           IconPtr   *icon1,
860                                           IconPtr   *icon2,
861                                           IconPtr   *overlayIcon1,
862                                           IconPtr   *overlayIcon2)
863  1 {
864  1   REST_FoundObjectPtr info;
                                 /* The real data from the client data */
865  1   u_hyper             size;     /* Size of the item */
866  1   time_t              theTime;  /* Date for the item */
867  1   time_t              now;      /* The current time */
868  1   BoolEnum            redraw;
                                 /* Flag whether or not to draw this cell */

870  1   /* Get the real pointer */
871  1   if ((column == 1) || (clientData != NULL))
872  2   {
873  2     info = (REST_FoundObjectPtr) clientData;
874  1   }
875  1   else
876  2   {
877  2     LBOX_GoCollRow (lbox, row);
878  2     info = (REST_FoundObjectPtr) LBOX_CurGetClientData (lbox);
879  2     LBOX_GoColRow (lbox, column, row);
880  1   }

882  1   /* Start with the default justification */
883  1   *justification = DRAW_JUSTLEFT | DRAW_JUSTVCENTER;

885  1   /* Never a second icon or overlay icon */
886  1   *icon2 = NULL;
887  1   *overlayIcon2 = NULL;
```



```
889  1   /* Initialize everything to blank */
890  1   *icon1 = NULL;
891  1   *overlayIcon1 = NULL;
892  1   STR_Cpy (text, "");

894  1   /* Determine each value based on the given info and the column */
895  1   if ((info != NULL) && (column <= NUMBER_FOUND_COLUMNS))
896  2   {
897  2     /* Draw this cell */
898  2     redraw = BOOL_TRUE;

900  2     if (column == BACKUP_TIME_COLUMN)
901  3     {
902  3       /* get the backup time string */
903  3       strftime (text,
904  3                 GMAX_CELL_STRING_LENGTH,
905  3                 "[%c]",
906  3                 localtime(&info->backupTime));
907  2     }

909  2     else if (column == MARK_COLUMN)
910  3     {
911  3       *icon1 = searchCheckBoxIcon;

913  3       /* Get the overlay icon */
914  3       if (GREST_IsObjectMarkedForTime (
                                GREST_Handle, info->object, info->backupTime))
915  4       {
916  4         if (LBOX_IsCurSelected (lbox))
917  4           *overlayIcon1 = searchSelCheckIcon;
918  4         else
919  4           *overlayIcon1 = searchCheckIcon;
920  3       }
921  3       else
922  4       {
923  4         *overlayIcon1 = NULL;
924  3       }
925  2     }
926  2     else if (column == ICON_COLUMN)
927  3     {
928  3       /* Get the type icon */
929  3       if (EDMRST_IsObjectContainer (GREST_Handle, info->object))
930  4       {
931  4         *icon1 = searchDirIcon;
932  3       }
933  3       else
934  4       {
935  4         *icon1 = searchFileIcon;
936  3       }

938  3       *overlayIcon1 = REST_GetStatusIcon (EDMRST_GetObjectStatus(
                                                              GREST_Handle,
939  3
                                                            info->object));
940  2     }

942  2     /* Column 3 is the Name column */
943  2     else if (column == NAME_COLUMN)
944  3     {
945  3       /* get the string for the full name */
946  3       STR_Cpy (text, EDMRST_GetObjectFullName (
                                                GREST_Handle, info->object));
947  2     }

949  2     /* Column 4 is the Name column */
950  2     else if (column == PERMISSIONS_COLUMN)
```

```
951  3      {
952  3        /* get the string for the permissions */
953  3        STR_Cpy (text, GREST_GetPermissionsString (
                                              GREST_Handle, info->object));
954  2      }

956  2      else if (column == OWNER_COLUMN)
957  3      {
958  3        /* get the string for the owner */
959  3        STR_Cpy (text, EDMRST_GetObjectOwnerString (
                                              GREST_Handle, info->object));
960  2      }

962  2      else if (column == GROUP_COLUMN)
963  3      {
964  3        /* get the string for the group */
965  3        STR_Cpy (text, EDMRST_GetObjectGroupString (
                                              GREST_Handle, info->object));
966  2      }

968  2      else if (column == SIZE_COLUMN)
969  3      {
970  3        /* get the string for the size */
971  3        size = EDMRST_GetObjectSize (GREST_Handle, info->object);
972  3        REST_SPrintHyper (text, size);
973  3        *justification = DRAW_JUSTRIGHT | DRAW_JUSTVCENTER;
974  2      }

976  2      else if (column == DATE_COLUMN)
977  3      {
978  3        /* get the string for the date */
979  3        theTime = EDMRST_GetObjectModDate (GREST_Handle, info->object);
980  3        strftime (text, GMAX_CELL_STRING_LENGTH, "%b %d", localtime (
                                                        &theTime));
981  2      }

983  2      else if (column == TIME_COLUMN)
984  3      {
985  3        /* get the string for the time */
986  3        theTime = EDMRST_GetObjectModDate (GREST_Handle, info->object);
987  3        now = time((time_t *)NULL);

989  3        /* If the object is over a year old, display the year */
990  3        if ((now - theTime) > SECONDS_PER_YEAR)
991  4        {
992  4          strftime (text, GMAX_CELL_STRING_LENGTH, "%y", localtime (
                                                        &theTime));
993  3        }
994  3        /* Otherwise display the time */
995  3        else
996  4        {
997  4          strftime (text, GMAX_CELL_STRING_LENGTH, "%H:%M", localtime (
                                                        &theTime));
998  3        }
999  2      }
1000 1    }
1001 1    else
1002 2    {
1003 2      /* Don't draw this cell */
1004 2      redraw = BOOL_TRUE;
1005 1    }


1008 1    /* Return whether or not to redraw */
1009 1    return (redraw);
1010   }
```



```
1012  /*****************************************************************
1013   * REST_SearchInProgress
1014   *
1015   * Description:
1016   *   This routine returns whether or not a search is currently in
                                                              progress.
1017   *
1018   * Parameters:
1019   *   None.
1020   *
1021   * Returns:
1022   *   BOOL_TRUE  - If a search has been started (
                                                  and is not yet finished)
1023   *   BOOL_FALSE - Otherwise.
1024   *
1025   *****************************************************************/

1027  BoolEnum  REST_SearchInProgress (void)
1028 1{
1029 1  return (searchInProgress);
1030  }
```

```
1032  /*********************************************************************
1033   * REST_SearchFindTimeout
1034   *
1035   * Description:
1036   *   This routine checks for the next set of find results
1037   *
1038   * Parameters:
1039   *   clientData - Not used
1040   *
1041   * Returns:
1042   *   None.
1043   *
1044   *********************************************************************/

1046  static void REST_SearchFindTimeout (ClientPtr clientData)
1047 1 {
1048 1   long          numEntries;                          /* Number found */
1049 1   Char          statusString[MEDIUM_STRING_LENGTH];
                                                  /* Search status string */
1050 1   GREST_Object  foundObjects[MAX_FOUND_OBJECTS];
                                                /* Array of objects found */
1051 1   time_t        times[MAX_FOUND_OBJECTS];
                                                /* Backup times for objects*/
1052 1   Int           i;                                   /* Loop index */
1053 1   eerrno_ty     eerrno;
                                                  /* Error code returned */
1054 1   BackupStatus  objectStatus;
                                                /* Current object's status */
1055 1   BoolEnum      isCancelled;
                                                  /* Flag to if cancelled */
1056 1   BoolEnum      displayError = BOOL_FALSE;
                                                 /* Flag to display error */
1057 1   BoolEnum      allowAll;
                                                /* Flag if all status is OK*/
1058 1   BoolEnum      allowGood;
                                                /* Flag if only Good is OK */
1059 1   BoolEnum      allowBad;
                                                 /* Flag if only bad is OK */

1061 1   /* Allocate space for the next array of objects */
1062 1   if (EDMRST_AllocRestorableObjects (GREST_Handle,
1063 1                                      foundObjects,
1064 1                                      MAX_FOUND_OBJECTS) == E_SUCCESS)
1065 2   {
1066 2     /* Initialize this pass */
1067 2     numEntries = 0;

1069 2     isCancelled = REST_CheckForCancel();

1071 2     if ((eerrno = EDMRST_GetFindResults (GREST_Handle,
1072 2                                          isCancelled,
1073 2                                          MAX_FOUND_OBJECTS,
1074 2                                          foundObjects,
1075 2                                          times,
1076 2                                          &numEntries,
1077 2                                          &REST_SearchCookie)) ==
                                                              E_SUCCESS)
1078 3     {
1079 3       /* Get the current status flags */
1080 3       allowAll  =TBUT_GetSelected((
                              TButPtr)REST_SearchWindow->AllStatusToggle);
1081 3       allowGood =TBUT_GetSelected((
                              TButPtr)REST_SearchWindow->GoodStatusToggle);
1082 3       allowBad  =TBUT_GetSelected((
                              TButPtr)REST_SearchWindow->BadStatusToggle);
```



```
1084 3       /* Update the number of entries found so far */
1085 3       REST_CurrentEntries += numEntries;

1087 3       /* Add each found object */
1088 3       for (i=0; i<numEntries; i++)
1089 4       {
1090 4         /* Make sure it passes our internal test for status (
                                                          not in API) */
1091 4         objectStatus =  EDMRST_GetObjectStatus (
                                          GREST_Handle, foundObjects[i]);
1092 4         if (allowAll ||
1093 4             (allowGood && (objectStatus == Backup_Good)) ||
1094 4             (allowBad && (objectStatus == Backup_Bad)))
1095 5         {
1096 5           /* Add the object to the list of found objects */
1097 5           REST_AddFoundItem (foundObjects[i], times[i]);
1098 4         }
1099 4         else
1100 5         {
1101 5           /* This one don't pass, decrement the number found */
1102 5           REST_CurrentEntries--;

1104 5           /* Free this object */
1105 5           EDMRST_FreeRestorableObjects (
                                      GREST_Handle, &foundObjects[i], 1);
1106 4         }
1107 3       }
1108 2     }
1109 2     else if (eerrno == EP_RB_RECOVER_FIND_INTERRUPTED)
1110 3     {
1111 3       /* The search was cancelled */
1112 3       GALERT_DisplayError ((WinPtr)REST_SearchWindow,
1113 3                            REST_GetErrorString (
                                          REST_CANCEL_SEARCH_TITLE),
1114 3                            GICON_GetError(),
1115 3                            REST_GetErrorString (
                                      REST_CANCEL_SEARCH_MESSAGE));

1117 3       /* We're done */
1118 3       REST_SearchCookie = DONE_COOKIE;
1119 2     }
1120 2     else if (eerrno != EP_RB_RECOVER_RPC_INCOMPLETE)
1121 3     {
1122 3       /* Flag that we want to display an error message */
1123 3       displayError = BOOL_TRUE;

1125 3       /* We're done */
1126 3       REST_SearchCookie = DONE_COOKIE;
1127 2     }

1129 2     /* Free up the left overs */
1130 2     if (numEntries < MAX_FOUND_OBJECTS)
1131 3     {
1132 3       EDMRST_FreeRestorableObjects (GREST_Handle,
1133 3                                     &foundObjects[numEntries],
1134 3                                     MAX_FOUND_OBJECTS - numEntries);
1135 2     }
1136 1   }
1137 1   else
1138 2   {
1139 2     /* Error allocating restorable objects, boogie on out */
1140 2     REST_SearchCookie = DONE_COOKIE;
1141 1   }

1143 1   if (REST_SearchCookie == DONE_COOKIE)
```

```
1144  2    {
1145  2      /* Flag that the search is done */
1146  2      searchInProgress = BOOL_FALSE;

1148  2      /* Make sure the window is still active */
1149  2      if (REST_SearchWindow != NULL)
1150  3      {
1151  3        /*
1152  3         * Update the search result status
1153  3         */

1155  3        /* Update the status text */
1156  3        STR_Sprintf (statusString, "%ld", REST_CurrentEntries);
1157  3        TED_SetStr ((
                      TEdPtr)REST_SearchWindow->ItemsFoundText, statusString);

1159  3        /* Fill the listbox */
1160  3        GUTIL_LBOX_FillVirtual (
                           REST_SearchWindow->FoundBox, REST_CurrentEntries);

1162  3        /* Update the search results action buttons */
1163  3        REST_SearchUpdateButtons ();

1165  3        /* Remove the working dialog */
1166  3        REST_SearchCancel ();

1168  3        /* If there was an error, display the error message */
1169  3        if (displayError)
1170  4        {
1171  4          /* Process events to remove the in progress dialog */
1172  4          EVENT_ProcessPending ();

1174  4          /* Display the current error message */
1175  4          REST_DisplayErrnoMessage ((WinPtr)REST_SearchWindow,
1176  4                                    REST_GetErrorString (
                                                 REST_SEARCH_FAIL_TITLE),
1177  4                                    NULL,
1178  4                                    eerrno);
1179  3        }
1180  2      }
1181  1    }
1182  1    else
1183  2    {
1184  2      if (eerrno == EP_RB_RECOVER_RPC_INCOMPLETE)
1185  3      {
1186  3        EVENT_StartCBackAlarm (REST_SearchFindTimeout,
1187  3                               clientData,
1188  3                               RESTORE_START_FIND_RESULTS_DELAY);
1189  2      }
1190  2      else
1191  3      {
1192  3        EVENT_StartCBackAlarm (REST_SearchFindTimeout,
1193  3                               clientData,
1194  3                               RESTORE_CONT_FIND_RESULTS_DELAY);
1195  2      }
1196  1    }
1197    }
```



```
1199  /*********************************************************************
1200   * Function Name:   REST_SearchStartSearch ()
1201   *
1202   * Description:
1203   *   This routine will start the search based on current criteria
1204   *   found in the widgets.
1205   *
1206   * Parameters:
1207   *   None.
1208   *
1209   * Success Outputs and Side Effects:
1210   *   The search results widgets will be updated.
1211   *
1212   * Returns:
1213   *   None.
1214   *
1215   *********************************************************************/

1217  void REST_SearchStartSearch (void)
1218  1 {
1219  1   EBREC_SearchCriteriaRec searchRec;
                                                  /* Criteria for search */
1220  1   Char           sizeString[SMALL_STRING_LENGTH];
                                                /* Size criteria from TEd */
1221  1   eerrno_ty      eerrno;
                                                  /* Error code returned */

1223  1   /* Do nothing if we're searching, or a restore is in progress */
1224  1   if (REST_SearchInProgress() || REST_RestoreInProgress())
1225  1     return;

1227  1   /* Get the search directory */
1228  1   TED_QueryStr ((TEdPtr)REST_SearchWindow->DirectoryText,
1229  1                 searchRec.startDirectory,
1230  1                 MAX_STRING_LENGTH);

1232  1   /* Standardize pathname: convert NT notation to UNIX */
1233  1   REST_StandardizePath(searchRec.startDirectory);

1235  1   if (STR_Cmp (searchRec.startDirectory, "") == CMP_EQUAL)
1236  1     STR_Cpy (searchRec.startDirectory, "/");

1238  1   /* Get whether or not to descend into sub directories */
1239  1   searchRec.descendDirectory = TBUT_GetSelected ((
                                  TButPtr)REST_SearchWindow->DescendToggle);

1241  1   /* Get the search string */
1242  1   TED_QueryStr ((
              TEdPtr)REST_SearchWindow->StringText, searchRec.searchString,
                                                    MAX_FILENAME_LENGTH);

1244  1   /* Standardize pathname: convert NT notation to UNIX */
1245  1   REST_StandardizePath(searchRec.searchString);

1247  1   if (STR_Cmp (searchRec.searchString, "") == CMP_EQUAL)
1248  2   {
1249  2     /* Default to everything, (
                                    exluding nothing = include everything) */
1250  2     STR_Cpy (searchRec.searchString, "*");
1251  2     searchRec.excludeString = BOOL_FALSE;
1252  1   }
1253  1   else
1254  2   {
1255  2     /* Get whether or not to exclude the search string */
1256  2     searchRec.excludeString = TBUT_GetSelected ((
```

```
                                TButPtr)REST_SearchWindow->ExcludeRadio);
1257 1   }

1259 1   /* Get the type of objects to search for */
1260 1   if (TBUT_GetSelected((TButPtr)REST_SearchWindow->AllTypesToggle))
1261 1     searchRec.typeOfFile = All_File_Types;
1262 1   else if (TBUT_GetSelected((
                              TButPtr)REST_SearchWindow->DirectoriesToggle))
1263 1     searchRec.typeOfFile = Directories_Only;
1264 1   else if (TBUT_GetSelected((TButPtr)REST_SearchWindow->FilesToggle))
1265 1     searchRec.typeOfFile = Files_Only;
1266 1   else
1267 1     searchRec.typeOfFile = Others_Only;

1269 1   /* Get the owner to limit the search to */
1270 1   TED_QueryStr ((
                    TEdPtr)REST_SearchWindow->OwnerText, searchRec.owner,
                                                    MAX_FILENAME_LENGTH);

1272 1   /* Get whether or not to exclude the owner */
1273 1   searchRec.excludeOwner = TBUT_GetSelected ((
                              TButPtr)REST_SearchWindow->ExcludeOwnerToggle);

1275 1   /* Get the group to limit the search to */
1276 1   TED_QueryStr ((
                    TEdPtr)REST_SearchWindow->GroupText, searchRec.group,
                                                    MAX_FILENAME_LENGTH);

1278 1   /* Get whether or not to exclude the group */
1279 1   searchRec.excludeGroup = TBUT_GetSelected ((
                              TButPtr)REST_SearchWindow->ExcludeGroupToggle);

1281 1   /* Get the size string */
1282 1   TED_QueryStr ((
TEdPtr)REST_SearchWindow->SizeText, sizeString, SMALL_STRING_LENGTH);
1283 1   if ((sizeString != NULL) && (STR_Len(sizeString) > 0))
1284 2   {
1285 2     /* Convert the string to a hyper */
1286 2     REST_ScanHyper (sizeString, &searchRec.sizeInBytes);

1288 2     /* Get the size criteria */
1289 2     if (TBUT_GetSelected((
                              TButPtr)REST_SearchWindow->GreaterThanToggle))
1290 2       searchRec.sizeMatch = EBREC_GreaterThan;
1291 2     else if (TBUT_GetSelected((
                             TButPtr)REST_SearchWindow->LessThanToggle))
1292 2       searchRec.sizeMatch = EBREC_LessThan;
1293 2     else
1294 2       searchRec.sizeMatch = EBREC_Equal;
1295 1   }
1296 1   else
1297 2   {
1298 2     /* Search for all sizes */
1299 2     searchRec.sizeMatch = EBREC_AllSizes;
1300 1   }

1302 1   /* Get the start backup date/time */
1303 1   searchRec.startTime = (time_t)WGT_GetClientRes ((
                               WgtPtr)REST_SearchWindow->StartDateOutput);

1305 1   /* Get the end backup date/time */
1306 1   searchRec.endTime = (time_t)WGT_GetClientRes ((
                                 WgtPtr)REST_SearchWindow->EndDateOutput);

1308 1   /* Flush the event queue */
1309 1   EVENT_DiscardUserEvents ();
```



```
1310 1   EVENT_ProcessPending ();

1312 1   /* Initialize the number of entries found so far */
1313 1   REST_CurrentEntries = 0;

1315 1   /* Flag that the search is in progress */
1316 1   searchInProgress = BOOL_TRUE;

1318 1   /* Clear the previous entries */
1319 1   REST_ClearFoundBox ();

1321 1   /* Find the next group of objects */
1322 1   if ((eerrno = EDMRST_FindRestorableObjects (
                                  GREST_Handle, &searchRec)) == E_SUCCESS)
1323 2   {
1324 2     REST_SearchCookie = INIT_COOKIE;
1325 2     EVENT_StartCBackAlarm (REST_SearchFindTimeout,
1326 2                            (ClientPtr)GREST_Handle,
1327 2                            RESTORE_START_FIND_RESULTS_DELAY);
1328 1   }
1329 1   else
1330 2   {
1331 2     /* Display the current error message */
1332 2     REST_DisplayErrnoMessage ((WinPtr)REST_SearchWindow,
1333 2                               REST_GetErrorString (
                                                 REST_SEARCH_FAIL_TITLE),
1334 2                               NULL,
1335 2                               eerrno);
1336 1   }
1337   }
```

```
1339  /*****************************************************************
1340   * Function Name:   REST_SearchToggleMark ()
1341   *
1342   * Description:
1343   *   This routine will mark/unmark the give item.
1344   *
1345   * Parameters:
1346   *   object (I) - The object to toggle the marking of
1347   *
1348   * Success Outputs and Side Effects:
1349   *   The object will be marked or unmarked for restore.
1350   *
1351   * Returns:
1352   *   None.
1353   *
1354   *****************************************************************/

1356  void REST_SearchToggleMark (ClientPtr object)
1357 1{
1358 1  REST_FoundObjectPtr   info;
                                          /* Next selected object */
1359 1  BoolEnum              marked = BOOL_FALSE;
                                        /* Flag if any marks worked */
1360 1  long                  numberMarked = 0;     /* Number of marks */
1361 1  long                  numberBad = 0;
                                          /* Number of bad marks */
1362 1  u_hyper               totalSize;            /* Total size marked */

1364 1  /* Do nothing if we're searching, or a restore is in progress */
1365 1  if (REST_SearchInProgress() || REST_RestoreInProgress())
1366 1    return;

1368 1  info = (REST_FoundObjectPtr)object;
1369 1  if (info != NULL)
1370 2  {
1371 2    /* Determine if this object is markable */
1372 2    if (GREST_IsObjectMarkableForTime (GREST_Handle,
1373 2                                       info->object,
1374 2                                       info->backupTime) &&
1375 2        ((EDMRST_GetObjectStatus (GREST_Handle,
1376 2                                  info->object) != Backup_Bad) ||
1377 2         REST_MarkBadFiles))
1378 3    {
1379 3      if (!GREST_IsObjectMarkedForTime (
                          GREST_Handle, info->object, info->backupTime))
1380 4      {
1381 4        marked = REST_MarkRestorableObject (info->object,
1382 4                                            info->backupTime,
1383 4                                            &numberMarked,
1384 4                                            &numberBad);
1385 3      }
1386 3      /* Otherwise, unmark it for restore */
1387 3      else
1388 4      {
1389 4        marked = REST_UnmarkRestorableObject (info->object,
1390 4                                              info->backupTime,
1391 4                                              &numberMarked,
1392 4                                              &numberBad);
1393 3      }

1395 3      /* Update the mark flags for all objects */
1396 3      REST_UpdateObjectMarks (currentWorkItemInfo);
1397 2    }

1399 2    if (marked)
```



```
1400 3    {
1401 3      /* Redraw the list boxes, to update the current marks */
1402 3      SAREA_RedrawParts ((SAreaPtr)REST_SearchWindow->FoundBox);
1403 3      SAREA_RedrawParts ((SAreaPtr)REST_RestoreWin->BackupListBox);
1404 3      SAREA_RedrawParts ((SAreaPtr)REST_RestoreWin->SelectedListBox);

1406 3      /* Update the mark information */
1407 3      EDMRST_GetMarkedTotalSize (GREST_Handle, &totalSize);
1408 3      REST_UpdateMarkedInfo (numberMarked,
1409 3                             totalSize,
1410 3                             numberBad);

1412 3      /* Update the search results action buttons */
1413 3      REST_SearchUpdateButtons ();
1414 2    }
1415 1  }
1416  }
```

```
1418  /*****************************************************************
1419   * Function Name:   REST_SearchSetMark ()
1420   *
1421   * Description:
1422   *   This routine will mark/unmark all selected items in the search
                                                                  results
1423   *   list box.
1424   *
1425   * Parameters:
1426   *   setMark (I) - Flag if the items should be marked or unmarked.
1427   *
1428   * Success Outputs and Side Effects:
1429   *   The selected items will be marked or unmarked for restore.
1430   *
1431   * Returns:
1432   *   None.
1433   *
1434   *****************************************************************/

1436  void REST_SearchSetMark (BoolEnum setMark)
1437 1 {
1438 1   REST_FoundObjectPtr   info;
                                          /* Next selected object */
1439 1   long                  numberMarked;      /* Number of marks */
1440 1   long                  numberBad;         /* Number of bad marks */
1441 1   BoolEnum              thisMark;
                                       /* Flag if this mark worked */
1442 1   BoolEnum              marked = BOOL_FALSE;
                                       /* Flag if any marks worked */
1443 1   u_hyper               totalSize;         /* Total marked size */

1445 1   /* Do nothing if we're searching, or a restore is in progress */
1446 1   if (REST_SearchInProgress() || REST_RestoreInProgress())
1447 1     return;

1449 1   /* If any rows are selected */
1450 1   if (LBOX_GoRowSelFirst (REST_SearchWindow->FoundBox) == BOOL_TRUE)
1451 2   {

1453 2     /* Go to the first row */
1454 2     LBOX_GoHome (REST_SearchWindow->FoundBox);

1456 2     /* Loop through all rows that have data */
1457 2     while ((info = (REST_FoundObjectPtr)LBOX_CurGetClientData(
                                  REST_SearchWindow->FoundBox)) != NULL)
1458 3     {
1459 3       /* If this row is selected, set the mark appropriately */
1460 3       if (LBOX_IsCurSelected (REST_SearchWindow->FoundBox))
1461 4       {
1462 4         /* If this is a mark operation, then mark it for restore */
1463 4         if (setMark)
1464 5         {
1465 5           if (!GREST_IsObjectMarkedForTime (
                          GREST_Handle, info->object, info->backupTime))
1466 6           {
1467 6             thisMark = REST_MarkRestorableObject (info->object,
1468 6                                                   info->backupTime,
1469 6                                                   &numberMarked,
1470 6                                                   &numberBad);
1471 5           }
1472 4         }
1473 4         /* Otherwise, unmark it for restore */
1474 4         else
1475 5         {
1476 5           if (GREST_IsObjectMarkedForTime (
```



```
                          GREST_Handle, info->object, info->backupTime))
1477 6           {
1478 6             thisMark = REST_UnmarkRestorableObject (info->object,
1479 6                                                     info->backupTime,
1480 6                                                     &numberMarked,
1481 6                                                     &numberBad);
1482 5           }
1483 4         }

1485 4         /* Set the flag if anything has been marked yet */
1486 4         marked = (marked || thisMark);
1487 3       }

1489 3       /* Go to the next row */
1490 3       LBOX_GoDown (REST_SearchWindow->FoundBox);
1491 2     }
1492 1   }

1494 1   /* Update the mark flags for all objects */
1495 1   REST_UpdateObjectMarks (currentWorkItemInfo);

1497 1   /* IF any mark or unmark was succesful, update the display */
1498 1   if (marked)
1499 2   {
1500 2     /* Redraw the list boxes, to update the current marks */
1501 2     SAREA_RedrawParts ((SAreaPtr)REST_SearchWindow->FoundBox);
1502 2     SAREA_RedrawParts ((SAreaPtr)REST_RestoreWin->BackupListBox);
1503 2     SAREA_RedrawParts ((SAreaPtr)REST_RestoreWin->SelectedListBox);

1505 2     /* Update the mark information */
1506 2     EDMRST_GetMarkedTotalSize (GREST_Handle, &totalSize);
1507 2     REST_UpdateMarkedInfo (numberMarked,
1508 2                            totalSize,
1509 2                            numberBad);
1510 1   }

1512 1   /* Update the search results action buttons */
1513 1   REST_SearchUpdateButtons ();
1514  }
```

```
1516  /*****************************************************************
1517   * Function Name:   REST_SearchSetView ()
1518   *
1519   * Description:
1520   *   This routine will set the current restore window's view to
1521   *   the currently selected item in the search results list box.
1522   *   This routine assumes only one item is selected, if multiples
1523   *   are selected, only the first selected item will be acted upon.
1524   *
1525   * Parameters:
1526   *   None.
1527   *
1528   * Success Outputs and Side Effects:
1529   *   The selected item will be in the restore view.
1530   *
1531   * Returns:
1532   *   None.
1533   *
1534   *****************************************************************/

1536  void REST_SearchSetView (void)
1537 1 {
1538 1   REST_FoundObjectPtr info;
                                        /* Info for the selected row */
1539 1   BoolEnum            found = BOOL_FALSE;
                                        /* Flag if we found the row */
1540 1   Str                 fullName;
                                         /* Full name of the object */

1542 1   /* Do nothing if we're searching, or a restore is in progress */
1543 1   if (REST_SearchInProgress() || REST_RestoreInProgress())
1544 1     return;

1546 1   /* If anything is selected */
1547 1   if (LBOX_GoRowSelFirst (REST_SearchWindow->FoundBox) == BOOL_TRUE)
1548 2   {
1549 2     /* Go to the first row */
1550 2     LBOX_GoHome (REST_SearchWindow->FoundBox);

1552 2     /* Loop through all rows that have data */
1553 2     info = (REST_FoundObjectPtr)LBOX_CurGetClientData(
                                          REST_SearchWindow->FoundBox);
1554 2     while ((!found) && (info != NULL))
1555 3     {
1556 3       /* If this row is selected, set the view */
1557 3       if (LBOX_IsCurSelected (REST_SearchWindow->FoundBox))
1558 4       {
1559 4         /* Set the backup view to this object */
1560 4         fullName = esl_strdup (EDMRST_GetObjectFullName (
                                          GREST_Handle, info->object));
1561 4         REST_StripDirectoryChars (fullName);

1563 4         REST_SetBackupView (info->backupTime, fullName);

1565 4         GUTIL_Free (fullName);

1567 4         /* Flag that we found it */
1568 4         found = BOOL_TRUE;
1569 3       }
1570 3       /* Otherwise, go to the next row */
1571 3       else
1572 4       {
1573 4         LBOX_GoDown (REST_SearchWindow->FoundBox);
1574 4         info = (REST_FoundObjectPtr)LBOX_CurGetClientData(
                                          REST_SearchWindow->FoundBox);
```



```
1575 3       }
1576 2     }
1577 1   }
1578  }
```

```
1580  /*********************************************************************
1581   * Function Name:   REST_ClearFoundBox ()
1582   *
1583   * Description:
1584   *   This routine will clear all items from the search results list
                                                                      box.
1585   *
1586   * Parameters:
1587   *   None.
1588   *
1589   * Success Outputs and Side Effects:
1590   *   The search results list box will be empty.
1591   *
1592   * Returns:
1593   *   None.
1594   *
1595   *********************************************************************/

1597  void REST_ClearFoundBox (void)
1598 1{
1599 1  REST_FoundObjectPtr thisObject;
1600 1  REST_FoundObjectPtr nextObject;

1602 1  /* Free the current found list */
1603 1  thisObject = firstFoundObject;
1604 1  while (thisObject != NULL)
1605 2  {
1606 2    nextObject = thisObject->next;
1607 2    REST_SearchDisposeInfo (thisObject);
1608 2    thisObject = nextObject;
1609 1  }
1610 1  firstFoundObject = NULL;
1611 1  lastFoundObject = NULL;

1613 1  /* Set everything to zero and blank */
1614 1  TED_SetStr ((TEdPtr)REST_SearchWindow->ItemsFoundText, "");

1616 1  /* Reset the list box and start at the beginning */
1617 1  LBOX_AllUnselect (REST_SearchWindow->FoundBox);
1618 1  LBOX_Reset (REST_SearchWindow->FoundBox);
1619 1  LBOX_GoHome (REST_SearchWindow->FoundBox);

1621 1  GUTIL_LBOX_FillVirtual (REST_SearchWindow->FoundBox, 0);

1623 1  /* Update the search results action buttons */
1624 1  REST_SearchUpdateButtons ();
1625  }
```



```
1627  /*********************************************************************
1628   * Function Name:   REST_SearchInitialize ()
1629   *
1630   * Description:
1631   *   This routine will initialize the use of the search window
                                                                   functions.
1632   *
1633   * Parameters:
1634   *   None.
1635   *
1636   * Success Outputs and Side Effects:
1637   *   None.
1638   *
1639   * Returns:
1640   *   None.
1641   *
1642   *********************************************************************/

1644  void REST_SearchInitialize (void)
1645 1{
1646 1  static BoolEnum initialized = BOOL_FALSE;
                                          /* Flag if we have init'd yet */

1648 1  /* If we haven't initialized already, initialize */
1649 1  if (!initialized)
1650 2  {

1652 2    /* Get the icons */
1653 2    searchDirIcon  = GICON_GetIconBySize (I_DIRCLOSED, ICON_SMALL);
1654 2    searchFileIcon = GICON_GetIconBySize (I_FILE, ICON_SMALL);
1655 2    searchCheckBoxIcon = GICON_GetIconBySize (I_CHECKBOX, ICON_SMALL);
1656 2    searchCheckIcon = GICON_GetIconBySize (I_CHECK, ICON_SMALL);
1657 2    searchSelCheckIcon = GICON_GetIconBySize (
                                          I_SELECTED_CHECK, ICON_SMALL);
1658 2    searchOffsiteIcon    = GICON_GetIconBySize (I_OFFSITE, ICON_SMALL);
1659 2    searchBadIcon = GICON_GetIcon (I_BADBUOBJECT);

1661 2    /* Flag that we have already initialized */
1662 2    initialized = BOOL_TRUE;
1663 1  }
1664  }
```

```
/*****************************************************************
 * restUtils.c
 *
 *
 * Copyright 1996 by Epoch Systems, Inc.
 *
 *
 * Mission Statement:
 *   This file contains the utility functions for the EDM Restore
                                                                window.
 *
 * Required includes:
 *   None
 *
 * Compile-Time Options:
 *   N/A
 *
 *
 * RCS Information:
 *     $RCSfile$
 *     $Revision$
 *     $Date$
 *****************************************************************/

#define  ERR_LIB  RESTORE

#include <esl/c_portable.h>
#include <esl/ep_xopen.h>

#include <stdlib.h>

#include <libgen.h>
#include <time.h>

#include <apppub.h>
#include <eventpub.h>
#include <rlibpub.h>
#include <gwpub.h>
#include <respub.h>
#include <strlpub.h>

#include "eerrno/e_errno.h"
#include <util/esl_string.h>
#include <restore/restore_api.h>
#include "restore/restMgr.h"
#define REST_UTIL_INIT
#include "restUtils.h"
#undef REST_UTIL_INIT
#include "restore.h"
#include "restoreP.h"
#include "restCBMgr.h"
#include "restAPIUtils.h"
#include "gutil/timeUtils.h"
#include "gutil/iconUtils.h"
#include "gutil/winUtils.h"
#include "gutil/msgUtils.h"
#include "gutil/miscUtils.h"
#include "gutil/fileMgr.h"
#include "gutil/guiDefines.h"
#include "gutil/guiUtils.h"
#include "gutil/alertMgr.h"

ERR_EXTERN
ERR_INMODULE("restore")

/*************
```



```
 * Constants *
 *************/

#define REST_MESSAGE_HEADER "RESTORE"

/*************
 * Globals   *
 *************/

static GUTIL_MsgHandle restMsgList = NULL;

/*****************************************************************
 * REST_UtilInitialize
 *
 * Description:
 *   This routine will initialize the routines in the utilities
                                                          portion of
 *   the restore functionality.
 *
 * Parameters:
 *   None.
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void REST_UtilInitialize (void)
{
  /* Initialize the message utilities */
  restMsgList = GUTIL_MsgHandleInit ("restore", "RestoreStrL");
}
```

```
 98  /*****************************************************************
 99   * REST_GetErrorString
100   *
101   * Description:
102   *   This routine will retrieve the error string for the given error
103   *   index.
104   *
105   * Parameters:
106   *   errorIndex (I) - Index of the error string to retrieve
107   *
108   * Returns:
109   *   Static pointer to the error string (Do not overwrite!)
110   *
111   *****************************************************************/

113  Str REST_GetErrorString (Int      errorIndex)
114 1{
115 1  Char  messageCode[SMALL_STRING_LENGTH];
                                    /* Built code in string format */

117 1  /* Build the message code using the given index */
118 1  STR_Sprintf (messageCode, "%s%d", REST_MESSAGE_HEADER, errorIndex);

120 1  /* Return the string at the index into the error message list */
121 1  return (GUTIL_MsgGetMsg (restMsgList, messageCode));
122  }
```



```
124  /*****************************************************************
125   * REST_DisplayErrnoMessage
126   *
127   * Description:
128   *   This routine will display the error dialog with the given mesage
129   *   and the text string for the given error code.  If the title given
130   *   is NULL, the title "Error" will be used.  If the errorText given
131   *   is NULL, only the eerrno error string will be displayed.
132   *
133   * Parameters:
134   *   title     (I) - Title of the error window (may be NULL)
135   *   errorText (I) - Text to display (may be NULL)
136   *   eerrno    (I) - The error code
137   *
138   * Returns:
139   *   None.
140   *
141   *****************************************************************/

143  void REST_DisplayErrnoMessage (WinPtr    parent,
144                                 Str       title,
145                                 Str       errorText,
146                                 eerrno_ty eerrno)
147 1{
148 1  Char       outputString[MAX_STRING_LENGTH];  /* String to display */
149 1  Char       tmpTitle[MAX_STRING_LENGTH];      /* Title to display */

151 1  /* Create the error text string to display */
152 1  if (errorText != NULL)
153 1    STR_Sprintf (outputString, "%s\n%s", errorText, e_get_error_text(
                                                          eerrno));
154 1  else
155 1    STR_Cpy (outputString, e_get_error_text(eerrno));

157 1  /* Create the title to display */
158 1  if (title == NULL)
159 1    STR_Cpy (tmpTitle, REST_GetErrorString (REST_ERROR_INDEX));
160 1  else
161 1    STR_Cpy (tmpTitle, title);

163 1  /* Display the error window */
164 1  GALERT_DisplayError (parent,
165 1                       tmpTitle,
166 1                       GICON_GetError(),
167 1                       outputString);
168  }
```

```
170  /*****************************************************************
171   * REST_SPrintHyper
172   *
173   * Description:
174   *   This routine will fill the given string with a textual
                                                          representation
175   *   of the given hyper.
176   *
177   * Parameters:
178   *   string (I) - The pre-allocated string to fill
179   *   size   (I) - The hyper to represent
180   *
181   * Returns:
182   *   None.
183   *
184   *****************************************************************/

186  void REST_SPrintHyper (Str     string,
187                         u_hyper size)
188 1{
189 1  /* Print the hyper to the string */
190 1  if (hyper_is_ulong(size))
191 1    STR_Sprintf (string, "%9u", size.low);
192 1  else
193 1    STR_Sprintf (string, "%s", format_u_hyper_for_output(size, 1, 0));
194  }
```



```
196  /*****************************************************************
197   * REST_ScanHyper
198   *
199   * Description:
200   *   This routine will retrieve a hyper from the given string.
201   *
202   * Parameters:
203   *   string (I) - The string to read
204   *   size   (I) - The hyper to read into
205   *
206   * Returns:
207   *   None.
208   *
209   *****************************************************************/

211  void REST_ScanHyper (Str      string,
212                       u_hyper *size)
213 1{
214 1  /* Call the esl routine to read the hyper */
215 1  string_to_u_hyper (string, size);
216  }
```

```
218  /*****************************************************************
219   * REST_GetGroupString
220   *
221   * Description:
222   *   This routine will return a string representation of the group
223   *   for the given info.
224   *
225   * Parameters:
226   *   info (I) - The item to get the group for.
227   *
228   * Returns:
229   *   A string representation of the group (Copy immediately)
230   *
231   *****************************************************************/

233  Str REST_GetGroupString (RestoreInfoPtr info)
234 1{
235 1  static Char  returnString[SMALL_STRING_LENGTH];
                                                  /* String to return */
236 1  Str          groupString;

238 1  if (info->restoreObject != NULL)
239 2  {
240 2    groupString = (Str)EDMRST_GetObjectGroupString (GREST_Handle,
241 2                                                    info->restoreObject);
242 2    if (groupString != NULL)
243 3    {
244 3      STR_Cpy (returnString, groupString);
245 2    }
246 2    else
247 3    {
248 3      STR_Cpy (returnString, "");
249 2    }
250 1  }
251 1  else
252 2  {
253 2    STR_Cpy (returnString, "");
254 1  }

256 1  return (returnString);
257  }
```



```
259  /*****************************************************************
260   * REST_GetOwnerString
261   *
262   * Description:
263   *   This routine will return a string representation of the Owner
264   *   for the given info.
265   *
266   * Parameters:
267   *   info (I) - The item to get the Owner for.
268   *
269   * Returns:
270   *   A string representation of the Owner (Copy immediately)
271   *
272   *****************************************************************/

274  Str REST_GetOwnerString (RestoreInfoPtr info)
275 1{
276 1  static Char  returnString[SMALL_STRING_LENGTH];
                                                  /* String to return */
277 1  Str          ownerString;

279 1  if (info->restoreObject != NULL)
280 2  {
281 2    ownerString = (Str)EDMRST_GetObjectOwnerString (GREST_Handle,
282 2                                                    info->restoreObject);
283 2    if (ownerString != NULL)
284 3    {
285 3      STR_Cpy (returnString, ownerString);
286 2    }
287 2    else
288 3    {
289 3      STR_Cpy (returnString, "");
290 2    }
291 1  }
292 1  else
293 2  {
294 2    STR_Cpy (returnString, "");
295 1  }

297 1  return (returnString);
298  }
```

```
300   /*****************************************************************
301    * REST_GetPermString
302    *
303    * Description:
304    *   This routine will return a string representation of the
                                                                Permissions
305    *   for the given info.
306    *
307    * Parameters:
308    *   info (I) - The item to get the Permissions for.
309    *
310    * Returns:
311    *   A string representation of the Permissions (Copy immediately)
312    *
313    *****************************************************************/

315   Str REST_GetPermString (RestoreInfoPtr info)
316 1 {
317 1   static Char  returnString[SMALL_STRING_LENGTH];
                                                  /* String to return */

318 1   Str          permString;

320 1   if (info->restoreObject != NULL)
321 2   {
322 2     permString = GREST_GetPermissionsString (
                                      GREST_Handle, info->restoreObject);
323 2     if (permString != NULL)
324 3     {
325 3       STR_Cpy (returnString, permString);
326 2     }
327 2     else
328 3     {
329 3       STR_Cpy (returnString, "");
330 2     }
331 1   }
332 1   else
333 2   {
334 2     STR_Cpy (returnString, "");
335 1   }

337 1   return (returnString);
338   }
```



```
340   /*****************************************************************
341    * REST_GetTimeDateString
342    *
343    * Description:
344    *   This routine will return a string representation of the time and
                                                                      date
345    *   for the given info.
346    *
347    * Parameters:
348    *   info (I) - The item to get the time and date for.
349    *
350    * Returns:
351    *   A string representation of the time and date (Copy immediately)
352    *
353    *****************************************************************/

355   Str REST_GetTimeDateString (time_t theTime)
356 1 {
357 1   static Char  returnString[MEDIUM_STRING_LENGTH];
                                                  /* String to return */


359 1   strftime (returnString,
360 1            MEDIUM_STRING_LENGTH,
361 1            "%m/%d/%y %H:%M",
362 1            localtime (&theTime));

364 1   return (returnString);
365   }
```

```
367  /*****************************************************************
368   * REST_GetSizeString
369   *
370   * Description:
371   *   This routine will return a string representation of the size
372   *   for the given info.
373   *
374   * Parameters:
375   *   info (I) - The item to get the size for.
376   *
377   * Returns:
378   *   A string representation of the size (Copy immediately)
379   *
380   *****************************************************************/

382  Str REST_GetSizeString (RestoreInfoPtr info)
383 1{
384 1  static Char  returnString[MEDIUM_STRING_LENGTH];
                                                /* String to return */
385 1  u_hyper      size;
                                               /* Size fo the object */

387 1  if ((info->type != REST_Client) &&
388 1      (info->type != REST_WorkItem) &&
389 1      (info->restoreObject != NULL))
390 2  {
391 2    size = EDMRST_GetObjectSize(GREST_Handle, info->restoreObject);
392 1  }
393 1  else
394 2  {
395 2    size.high = 0;
396 2    size.low = 0;
397 1  }

399 1  REST_SPrintHyper (returnString, size);

401 1  return (returnString);
402  }
```



```
404  /*****************************************************************
405   * REST_GetDateString
406   *
407   * Description:
408   *   This routine will return a string representation of the Date
409   *   for the given info.
410   *
411   * Parameters:
412   *   info (I) - The item to get the Date for.
413   *
414   * Returns:
415   *   A string representation of the Date (Copy immediately)
416   *
417   *****************************************************************/

419  Str REST_GetDateString (RestoreInfoPtr info)
420 1{
421 1  static Char  returnString[MEDIUM_STRING_LENGTH];
                                                /* String to return */
422 1  time_t       theTime;
                                               /* Time of the object */

424 1  if (info->restoreObject != NULL)
425 2  {
426 2    theTime = EDMRST_GetObjectModDate(
                                        GREST_Handle, info->restoreObject);
427 2    strftime (returnString,
428 2              MEDIUM_STRING_LENGTH,
429 2              "%b %d",
430 2              localtime (&theTime));
431 1  }
432 1  else
433 2  {
434 2    STR_Cpy (returnString, "");
435 1  }
436 1  return (returnString);
437  }
```

```
439  /*****************************************************************
440   * REST_GetTimeString
441   *
442   * Description:
443   *   This routine will return a string representation of the Time
444   *   for the given info.
445   *
446   * Parameters:
447   *   info (I) - The item to get the Time for.
448   *
449   * Returns:
450   *   A string representation of the Time (Copy immediately)
451   *
452   *****************************************************************/

454  Str REST_GetTimeString (RestoreInfoPtr info)
455 1 {
456 1   time_t       modificationTime;
                                          /* Time of the object */
457 1   time_t       now;                      /* Current time */
458 1   static Char  returnString[MEDIUM_STRING_LENGTH];
                                          /* String to return */

460 1   if (info->restoreObject != NULL)
461 2   {
462 2     /* Get the modification time */
463 2     modificationTime = EDMRST_GetObjectModDate (GREST_Handle,
464 2                                                 info->restoreObject);

466 2     /* If the time is more than 6 months old, show the year only */
467 2     now = time((time_t *)NULL);
468 2     if ((now - modificationTime) > (SECONDS_PER_YEAR / 2))
469 3     {
470 3       strftime (returnString,
471 3                 MEDIUM_STRING_LENGTH,
472 3                 "%Y",
473 3                 localtime (&modificationTime));
474 2     }
475 2     /* Otherwise, show the time */
476 2     else
477 3     {
478 3       strftime (returnString,
479 3                 MEDIUM_STRING_LENGTH,
480 3                 "%H:%M",
481 3                 localtime (&modificationTime));
482 2     }
483 1   }
484 1   else
485 2   {
486 2     STR_Cpy (returnString, "");
487 1   }
488 1   return (returnString);
489  }
```



```
491  /*****************************************************************
492   * REST_StripDirectoryChars
493   *
494   * Description:
495   *   This routine will strip off the ending directory characters from
496   *   the given string;
497   *
498   * Parameters:
499   *   dirString (I) - The string to strip
500   *
501   * Returns:
502   *   None.
503   *
504   *****************************************************************/

506  void REST_StripDirectoryChars (Str dirString)
507 1 {
508 1   BoolEnum    lastCharFound = BOOL_FALSE;
                                          /* Flag if we verified last char */
509 1   Int         lastChar;
                                          /* Current position of last char */

511 1   /*
512 1    * Strip off trailing spaces and directory markers
513 1    */

515 1   lastChar = STR_Len (dirString) - 1;
516 1   while ((!lastCharFound) && (lastChar > 0))
517 2   {
518 2     /* Check if the last character is a blank or directory marker */
519 2     if ((dirString[lastChar] == ' ') ||
520 2         (dirString[lastChar] == '/') ||
521 2         (dirString[lastChar] == '\\'))
522 3     {
523 3       /* Remove the character from the string */
524 3       dirString[lastChar] = '\0';
525 3       lastChar--;
526 2     }
527 2     else
528 3     {
529 3       /* Valid character */
530 3       lastCharFound = BOOL_TRUE;
531 2     }
532 1   }
533  }
```

```
535  /*****************************************************************
536   * REST_GetFullName
537   *
538   * Description:
539   *   This routine will return a string representation of the FullName
540   *   for the given info.
541   *
542   * Parameters:
543   *   info (I) - The item to get the FullName for.
544   *
545   * Returns:
546   *   A string representation of the FullName (Copy immediately)
547   *
548   *****************************************************************/
549
550  Str REST_GetFullName (RestoreInfoPtr info)
551 1{
552 1  static Str  returnString = NULL;          /* The string to return */
553
554 1  /* Free the old string if necessary */
555 1  if (returnString != NULL)
556 1    GUTIL_Free (returnString);
557
558 1  /* If this is a client, the full name is the name */
559 1  if (info->type == REST_Client)
560 2  {
561 2    returnString = esl_strdup (info->name);
562 1  }
563
564 1  /* If this is a restore object,
                              get the full name from the Restore API */
565 1  else if (info->restoreObject != NULL)
566 2  {
567 2    returnString = esl_strdup (EDMRST_GetObjectFullName (
                                    GREST_Handle, info->restoreObject));
568 1  }
569
570 1  /* Else, just return the name (this should never happen) */
571 1  else
572 2  {
573 2    returnString = esl_strdup (info->name);
574 1  }
575
576 1  /* Strip off any trailing directory characters */
577 1  if ((info->type != REST_Client)    &&
578 1      (info->type != REST_WorkItem)  &&
579 1      (info->type != REST_FailedWorkItem))
580 2  {
581 2    REST_StripDirectoryChars (returnString);
582 1  }
583
584 1  return (returnString);
585 }
```



```
587  /*****************************************************************
588   * REST_GetNameString
589   *
590   * Description:
591   *   This routine will return a string representation of the Name
592   *   for the given info.
593   *
594   * Parameters:
595   *   fileObject (I) - The item to get the Name for.
596   *
597   * Returns:
598   *   A string representation of the Name (Copy immediately)
599   *
600   *****************************************************************/
601
602  Str REST_GetNameString (GFMGR_Context fMgr,
603                          GFMGR_Object  fileObject)
604 1{
605 1  RestoreInfoPtr  info; /* Real info for the object */
606
607 1  info = (RestoreInfoPtr)fileObject;
608
609 1  return (info->name);
610 }
```

```
612   /*****************************************************************
613    * REST_IsParentString
614    *
615    * Description:
616    *   This routine will return whether or not the given parent is the
617    *   parent of the given full child name.
618    *
619    * Parameters:
620    *   parent    (I) - The parent to check.
621    *   childName (I) - The full name of the child looking for.
622    *
623    * Returns:
624    *   BOOL_TRUE  - if it is the parent.
625    *   BOOL_FALSE - if it is not the parent.
626    *
627    *****************************************************************/

629   BoolEnum  REST_IsParentString (RestoreInfoPtr parent,
630                                  Str            childName)
631 1 {
632 1   Str      fullParentName; /* Full path for parent */
633 1   Int      length;
                                 /* Length of the full path name for parent */
634 1   BoolEnum retVal;         /* Value to return */

636 1   /* Get the full path name for the parent object */
637 1   fullParentName = esl_strdup (REST_GetFullName(parent));
638 1   length = STR_Len(fullParentName);

640 1   /* If the parent name ends in a directory marker,
                                                 compare only before it */
641 1   if ((fullParentName[length-1] == '\\') ||
642 1       (fullParentName[length-1] == '/'))
643 2   {
644 2     /* Decrease our comparison by a character */
645 2     length--;
646 1   }

648 1   /*
649 1    * Check that they match up to the end of the parent name
650 1    * and then the next child character is the dir symbol ('/')
651 1    * or the end of the string (files are children of themselves).
652 1    */

654 1   if (STR_NCmp (fullParentName, childName, length) == CMP_EQUAL)
655 2   {
656 2     retVal = ((childName[length] == '\0') ||
657 2               (childName[length] == '\\') ||
658 2               (childName[length] == '/'));
659 1   }
660 1   else
661 2   {
662 2     retVal = BOOL_FALSE;
663 1   }

665 1   /* Now free the full name */
666 1   GUTIL_Free (fullParentName);

668 1   /* return */
669 1   return (retVal);
670   }
```



```
672   /*****************************************************************
673    * REST_GetStatusIcon
674    *
675    * Description:
676    *   This routine will return the status overlay icon for the given
677    *   restore object status.
678    *
679    * Parameters:
680    *   status (I) - The status of the object.
681    *
682    * Returns:
683    *   IconPtr - The icon to display as the overlay icon, possibly NULL
684    *
685    *****************************************************************/

687   IconPtr  REST_GetStatusIcon (BackupStatus status)
688 1 {
689 1   IconPtr returnIcon;

691 1   /* If expired, return the expired icon */
692 1   if (status == Backup_Expired)
693 2   {
694 2     returnIcon = REST_ExpiredIcon;
695 1   }

697 1   /* Else If expired children, return the missing children icon */
698 1   else if (status == Backup_Child_Without_Data)
699 2   {
700 2     returnIcon = REST_MissingChildrenIcon;
701 1   }

703 1   /* Else If the object status is bad, return the bad icon */
704 1   else if (status == Backup_Bad)
705 2   {
706 2     returnIcon = REST_BadIcon;
707 1   }

709 1   /* Else return No icon */
710 1   else
711 2   {
712 2     returnIcon = NULL;
713 1   }

715 1   return (returnIcon);
716   }
```

```
718  /*****************************************************************
719   * REST_GetIcon
720   *
721   * Description:
722   *   This routine will return the icon to display for the given object.
723   *   This routine is generally called from the file manager.
724   *
725   * Parameters:
726   *   fileObject (I) - The object
727   *   isOpen     (I) - Flag if the object is "open"
728   *
729   * Returns:
730   *   The icon to display (can be NULL).
731   *
732   *****************************************************************/

734  IconPtr REST_GetIcon  (GFMGR_Context fMgr,
735                         GFMGR_Object  fileObject,
736                         BoolEnum      isOpen)
737 1{
738 1  RestoreInfoPtr  info;         /* The real info for the object */
739 1  IconPtr         returnIcon;   /* The icon to display */

741 1  /* Cast back to the real object */
742 1  info = (RestoreInfoPtr)fileObject;

744 1  /* Get the status icon */
745 1  returnIcon = REST_GetStatusIcon (info->status);

747 1  /* If status does not need to be displayed,
                                          display the object icon */
748 1  if (returnIcon == NULL)
749 2  {
750 2    /* If this object is open and it is a directory,
                                         return the dir open icon */
751 2    if ((isOpen) && (info->type == REST_Directory))
752 2      returnIcon = REST_DirOpenIcon;

754 2    /* Else return the current icon for the object */
755 2    else
756 2      returnIcon = info->icon;
757 1  }

759 1  /* Return the icon */
760 1  return (returnIcon);
761  }
```



```
763  /*****************************************************************
764   * REST_GetIcon2
765   *
766   * Description:
767   *   This routine will return the icon to display for the given object.
768   *   This routine is generally called from the file manager.
769   *
770   * Parameters:
771   *   fileObject (I) - The object
772   *   isOpen     (I) - Flag if the object is "open"
773   *
774   * Returns:
775   *   The icon to display (can be NULL).
776   *
777   *****************************************************************/

779  IconPtr REST_GetIcon2 (GFMGR_Context fMgr,
780                         GFMGR_Object  fileObject,
781                         BoolEnum      isOpen)
782 1{
783 1  RestoreInfoPtr  info;         /* The real info for the object */
784 1  IconPtr         tempIcon;     /* Temporary Icon */
785 1  IconPtr         returnIcon;   /* The icon to display */

787 1  /* Cast back to the real object */
788 1  info = (RestoreInfoPtr)fileObject;

790 1  tempIcon = REST_GetStatusIcon (info->status);

792 1  /* If status needed to be displayed,
                                         display the object icon second */
793 1  if (tempIcon != NULL)
794 2  {
795 2    /* If this object is open and it is a directory,
                                              return the dir open icon */
796 2    if ((isOpen) && (info->type == REST_Directory))
797 2      returnIcon = REST_DirOpenIcon;

799 2    /* Else return the current icon for the object */
800 2    else
801 2      returnIcon = info->icon;
802 1  }
803 1  else
804 2  {
805 2    returnIcon = NULL;
806 1  }

808 1  /* Return the icon */
809 1  return (returnIcon);
810  }
```

```
812  /*****************************************************************
813   * REST_GetOverlayIcon
814   *
815   * Description:
816   *   This routine will return the overlay icon to display for the
                                                              given object.
817   *   This routine is generally called from the file manager.
818   *
819   * Parameters:
820   *   fileObject (I) - The object
821   *
822   * Returns:
823   *   The overlay icon to display (can be NULL).
824   *
825   *****************************************************************/

827  IconPtr REST_GetOverlayIcon  (GFMGR_Context fMgr,
828                                GFMGR_Object  fileObject)
829 1 {
830 1   return (NULL);
831  }
```



```
833  /*****************************************************************
834   * REST_GetOverlayIcon2
835   *
836   * Description:
837   *   This routine will return the overlay icon to display for the
                                                              given object.
838   *   This routine is generally called from the file manager.
839   *
840   * Parameters:
841   *   fileObject (I) - The object
842   *
843   * Returns:
844   *   The overlay icon to display (can be NULL).
845   *
846   *****************************************************************/

848  IconPtr REST_GetOverlayIcon2 (GFMGR_Context fMgr,
849                                GFMGR_Object  fileObject)
850 1 {
851 1   return (NULL);
852  }
```

```
854  /*****************************************************************
855   * REST_DisposeInfo
856   *
857   * Description:
858   *   This routine will free the info memory associated with the given
                                                                    object.
859   *
860   * Parameters:
861   *   info       (I) - The item to dispose of.
862   *
863   * Returns:
864   *   None.
865   *
866   *****************************************************************/

868  void REST_DisposeInfo (RestoreInfoPtr info)
869 1 {

871 1   /* Sanity check, make sure no bonehead called us with a NULL info */
872 1   if (info != NULL)
873 2   {

875 2     /* Free up the name for this obect */
876 2     GUTIL_Free (info->name);

878 2     /* Free up the error string if it existed */
879 2     if (info->errorString != NULL)
880 2       GUTIL_Free (info->errorString);

882 2     /* Free up the object */
883 2     GUTIL_Free (info);
884 1   }
885  }
```



```
887  /*****************************************************************
888   * REST_FreeInfo
889   *
890   * Description:
891   *   This routine will free all memory associated with the given
                                                                    object
892   *   and all of the object's children.
893   *
894   * Parameters:
895   *   info       (I) - The item to free.
896   *
897   * Returns:
898   *   None.
899   *
900   *****************************************************************/

902  void REST_FreeInfo (RestoreInfoPtr info)
903 1 {
904 1   RestoreInfoPtr  childInfo;  /* Child info to free */
905 1   RestoreInfoPtr  nextChild;  /* Pointer to next child */

907 1   /* Make sure there is info to free */
908 1   if (info != NULL)
909 2   {

911 2     /* Free up any children of this object */
912 2     childInfo = info->children;
913 2     while (childInfo != NULL)
914 3     {
915 3       nextChild = childInfo->next;
916 3       REST_FreeInfo (childInfo);
917 3       childInfo = nextChild;
918 2     }

920 2     /* Free up the associated restorable object if it exists */
921 2     if (info->restoreObject != NULL)
922 3     {
923 3       EDMRST_FreeRestorableObjects (
                                    GREST_Handle, &info->restoreObject, 1);
924 2     }

926 2     /* Free up the memory for this obect */
927 2     REST_DisposeInfo (info);
928 1   }
929  }
```

```
931   /*********************************************************************
932    * REST_IsLessThan
933    *
934    * Description:
935    *   This routine determins if an object is "less than" another object
936    *   based on current sort criteria.
937    *
938    * Parameters:
939    *   item1 (I) - The item in question
940    *   item2 (I) - The item to compare to
941    *
942    * Returns:
943    *   BOOL_TRUE  - If item1 < item2
944    *   BOOL_FALSE - If item1 >= item2
945    *
946    *********************************************************************/

948   BoolEnum REST_IsLessThan (RestoreInfoPtr item1,
949                             RestoreInfoPtr item2)
950 1 {
951 1   Char           string1[GMAX_OBJECT_LENGTH]; /* String for item1 */
952 1   Char           string2[GMAX_OBJECT_LENGTH]; /* String for item2 */
953 1   u_hyper        size1;                       /* Size for item1 */
954 1   u_hyper        size2;                       /* Size for item2 */
955 1   CmpEnum        cmpValue;
                                        /* Value returned by comparison */
956 1   BoolEnum       returnValue;                 /* Value to return */
957 1   REST_SortType  currentSort;
                                         /* Current type of sort to use */

959 1   /* Based on the current sort criteria, compare the two items */
960 1   currentSort = REST_GetSort ();
961 1   switch (currentSort)
962 2   {

964 2     /* If sorting by name, compare the names */
965 2     case REST_ByName:
966 2       returnValue = (STR_Cmp (item1->name, item2->name) == CMP_UNDER);
967 2       break;

969 2     /* If sorting by type, compare the types */
970 2     case REST_ByType:
971 2       /* If they are the same type, sort by name */
972 2       if (item1->type == item2->type)
973 2         returnValue = (STR_Cmp (
                                  item1->name, item2->name) == CMP_UNDER);
974 2       else
975 2         returnValue = (item1->type < item2->type);
976 2       break;

978 2     /* If sorting by date, compare the dates */
979 2     case REST_ByDate:
980 2       /* Date works backwards, sort by latest time first */
981 2       returnValue = (EDMRST_GetObjectModDate (GREST_Handle,
982 2                                               item1->restoreObject) >
983 2                      EDMRST_GetObjectModDate (GREST_Handle,
984 2                                               item2->restoreObject));
985 2       break;

987 2     /* If sorting by size, compare the sizes */
988 2     case REST_BySize:
989 2       /* Size works backwards, sort by largest size first */
990 2       size1 = EDMRST_GetObjectSize(GREST_Handle, item1->restoreObject);
991 2       size2 = EDMRST_GetObjectSize(GREST_Handle, item2->restoreObject);
992 2       if (size1.high > size2.high)
```



```
993 2         returnValue = BOOL_TRUE;
994 2       else if ((size1.high == size2.high) && (size1.low > size2.low))
995 2         returnValue = BOOL_TRUE;
996 2       else if ((size1.high == size2.high) && (size1.low == size2.low))
997 2         returnValue = (STR_Cmp (
                                    item1->name, item2->name) == CMP_UNDER);
998 2       else
999 2         returnValue = BOOL_FALSE;
1000 2      break;

1002 2    /* If sorting by owner, compare the owners */
1003 2    case REST_ByOwner:
1004 2      STR_Cpy (string1, REST_GetOwnerString(item1));
1005 2      STR_Cpy (string2, REST_GetOwnerString(item2));
1006 2      cmpValue = STR_Cmp (string1, string2);
1007 2      if (cmpValue == CMP_UNDER)
1008 2        returnValue = BOOL_TRUE;
1009 2      else if (cmpValue == CMP_EQUAL)
1010 2        returnValue = (STR_Cmp (
                                    item1->name, item2->name) == CMP_UNDER);
1011 2      else
1012 2        returnValue = BOOL_FALSE;
1013 2      break;

1015 2    default:
1016 2      /* Hmmmm... Oh well, sort by name. */
1017 2      returnValue = (STR_Cmp (item1->name, item2->name) == CMP_UNDER);
1018 1  }

1020 1  /* Return the value determined */
1021 1  return (returnValue);
1022   }
```

```
1024  /*********************************************************************
1025   * REST_IsBadObject
1026   *
1027   * Description:
1028   *   This routine will determine if the object status is bad.
1029   *
1030   * Parameters:
1031   *   info   (I) - The object to check the status of.
1032   *
1033   * Returns:
1034   *   BOOL_TRUE  - If the object status is bad
1035   *   BOOL_FALSE - otherwise
1036   *
1037   *********************************************************************/

1039  BoolEnum REST_IsBadObject (RestoreInfoPtr info)
1040 1{
1041 1  BoolEnum returnValue;  /* Value to return */

1043 1  /* If this is a Restore API object and it has a bad status */
1044 1  if (info->status == Backup_Bad)
1045 2  {
1046 2    /* This is a bad, bad, object */
1047 2    returnValue = BOOL_TRUE;
1048 1  }
1049 1  else
1050 2  {
1051 2    /* This object has been very good */
1052 2    returnValue = BOOL_FALSE;
1053 1  }

1055 1  /* Return whether or not the object is bad */
1056 1  return (returnValue);
1057  }
```



```
1059  /*********************************************************************
1060   * REST_HasChildren
1061   *
1062   * Description:
1063   *   This routine is provided for the file manager.  It determines
1064   *   if the given object can have children.
1065   *
1066   * Parameters:
1067   *   fileObject (I) - Object to check for children.
1068   *
1069   * Returns:
1070   *   None.
1071   *
1072   *********************************************************************/

1074  BoolEnum REST_HasChildren (GFMGR_Context fMgr,
1075                             GFMGR_Object  fileObject)
1076 1{
1077 1  RestoreInfoPtr   info;
                                /* Real representation of the object */
1078 1  BoolEnum         returnValue; /* Value to return */

1080 1  /* Get the real data type for the object */
1081 1  info = (RestoreInfoPtr)fileObject;

1083 1  /* If this is a directory, client, or a work-item,
                                                    it has children */
1084 1  if ((info->type == REST_Directory) ||
1085 1      (info->type == REST_Client)    ||
1086 1      (info->type == REST_WorkItem)  ||
1087 1      (info->type == REST_FailedWorkItem))
1088 2  {
1089 2    returnValue = BOOL_TRUE;
1090 1  }

1092 1  /* Otherwise it has no children */
1093 1  else
1094 2  {
1095 2    returnValue = BOOL_FALSE;
1096 1  }

1098 1  /* Return whether or not it has children */
1099 1  return (returnValue);
1100  }
```

```
1102  /*****************************************************************
1103   * REST_StandardizePath
1104   *
1105   * Description:
1106   *    This routine is provided to convert the given path into
1107   *    standard UNIX format. i.e., C:\ABC will be modified as /C/ABC
1108   *    All backslashes will be made forwardslashes if a colon appears
1109   *    after the first Alphabet.
1110   *
1111   * Parameters:
1112   *    Str : The data in this string is modified.
1113   *
1114   * Returns:
1115   *    BOOL_TRUE
1116   *
1117   *****************************************************************/

1119  BoolEnum REST_StandardizePath(Str pathname)
1120  1 {
1121  1    int i;
1122  1    int pathlength = strlen(pathname);

1124  1    if(pathname == NULL || pathlength < 2 )
1125  1        return BOOL_FALSE;

1127  1    /* NT style paths have first character as a drive name, i.e.,
                                                                  a single
1128  1       character followed by a colon */

1130  1    if( ':' == pathname[1]  &&
1131  1                ( toupper(pathname[0]) >= 'A' && toupper(
                                                  pathname[0]) <= 'Z' ))
1132  2    {
1133  2       pathname[1] = pathname[0];
1134  2       pathname[0] = '/';
1135  1    }

1137  1    /* If it is NT style path,
                            then the backslashes should also be converted
1138  1     * into forward slashes.
1139  1     */
1140  1     for(i=2; i<pathlength; i++ )
1141  2     {
1142  2        if( '\\' == pathname[i] )     /* need another back slash to hide
                                                                  it */
1143  2            pathname[i] = '/';
1144  1     }
1145  1     return BOOL_TRUE;
1146   }
```

```
/*****************************************************************
 * restStartMgr.c
 *
 *
 * Copyright 1996 by Epoch Systems, Inc.
 *
 *
 * Mission Statement:
 *   This file contains the functions necessary for starting the actual
 *   the EDM Restoral.
 *
 * Required includes:
 *   None
 *
 * Compile-Time Options:
 *   N/A
 *
 *
 * RCS Information:
 *     $RCSfile$
 *     $Revision$
 *     $Date$
 *****************************************************************/

#define  ERR_LIB  RESTORE

#include <esl/c_portable.h>
#include <esl/ep_xopen.h>
#include <util/esl_core.h>
#include <eerrno/e_errno.h>
#include <util/esl_string.h>

#include <stdlib.h>

/* WARNING: UNIX DEPENDENCY!!! Used for polling sockets */
#include <stropts.h>
#include <poll.h>
#include <unistd.h>

#include <libgen.h>
#include <time.h>

#include <apppub.h>
#include <eventpub.h>
#include <rlibpub.h>
#include <gwpub.h>
#include <respub.h>
#include <lboxpub.h>
#include <tbutpub.h>
#include <tedpub.h>
#include <winpub.h>
#include <strlpub.h>
#include <drawpub.h>
#include <dsplypub.h>

#include "eerrno/e_errno.h"
#include "fbrowser/epcomm_api.h"
#include "util/hyper.h"

#include <restore/restore_api.h>
#include "restore/restMgr.h"
#include "restoreP.h"
#include "restUtils.h"
#include "restSearch.h"
#include "restSelMgr.h"
```



```
#include "restCBMgr.h"
#include "restDest.h"
#include "restProgress.h"
#include "restQuestion.h"
#include "gutil/timeUtils.h"
#include "gutil/winUtils.h"
#include "gutil/guiDefines.h"
#include "gutil/guiUtils.h"
#include "gutil/iconDefs.h"
#include "gutil/iconUtils.h"
#include "gutil/alertMgr.h"

/************************
 * Local Data Structures *
 ************************/

typedef struct _REST_TimerRec
{
  int    startFd;          /* The File descriptor to get data from */
  void   (*auxprocRead)();
                           /* The routine to call when data is received */
  void   *data;            /* Data to pass on to the read routine */
} REST_TimerRec, *REST_TimerPtr;

/*****************************************************************
 * REST_VerifyMedia
 *
 * Description:
 *   This routine will verify the media for the current restore
 *
 * Parameters:
 *   None.
 *
 * Returns:
 *   BOOL_TRUE  - If it is OK to proceed with restore
 *   BOOL_FALSE - If there is offline/offsite media and the user
                                                          cancelled
 *
 *****************************************************************/

static BoolEnum REST_VerifyMedia ( void )
{
  GREST_MediaObject nextMedia;
                                          /* Next media object in list */
  BoolEnum          OKtoRestore = BOOL_TRUE;
                                            /* Flag if OK to proceed */
  BoolEnum          errorDisplayed = BOOL_FALSE;
                                          /* Flag if we already asked */


  /*
   * Loop through the current media list,
   * If any one is not online,
                                  ask the user if he/she wants to proceed.
   */

  LBOX_GoCol1Row (REST_RestoreWin->MediaListBox, 1);
  nextMedia = (GREST_MediaObject) LBOX_CurGetClientData (
                                          REST_RestoreWin->MediaListBox);
  while ((nextMedia != NULL) && (!errorDisplayed))
  {
    /* IF the media is offline or offsite, display a warning */
    if (EDMRST_GetMediaStatus (
                                  GREST_Handle, nextMedia) != Media_Online)
    {

      /* Flag that the error has been displayed */
```

```
124 3       errorDisplayed = BOOL_TRUE;

126 3       /* Ask the user if he/she wants to continue anyways */
127 3       if (GALERT_DisplayQuestion ((WinPtr)REST_RestoreWin,
128 3                                   REST_GetErrorString(
                                            REST_OFFLINE_MEDIA_TITLE),
129 3                                   GICON_GetWarning(),
130 3                                   REST_GetErrorString(
                                            REST_OFFLINE_MEDIA_ERROR),
131 3                                   BOOL_FALSE) != GALERT_Affirmative)

133 4       {
134 4         /* The user cancelled the restore */
135 4         OKtoRestore = BOOL_FALSE;
136 3       }
137 2     }
138 2     LBOX_GoDown (REST_RestoreWin->MediaListBox);
139 2     nextMedia = (GREST_MediaObject) LBOX_CurGetClientData (
                                          REST_RestoreWin->MediaListBox);
140 1   }

142 1   /* Return whether or not it is OK to proceed with the restore */
143 1   return (OKtoRestore);
144   }
```



```
146 /*****************************************************************
147  * REST_StartRestoral
148  *
149  * Description:
150  *   This routine start the restoral of the currently marked objects.
151  *
152  * Parameters:
153  *   None.
154  *
155  * Returns:
156  *   None.
157  *
158  *****************************************************************/

160 void REST_StartRestoral (void)
161 1 {
162 1   BoolEnum          inPlace;
                                        /* Flag if this is inplace */
163 1   Char              hostName[MAX_CLIENT_NAME_LENGTH];
                                        /* Host to restore to */
164 1   Char              destHostName[MAX_CLIENT_NAME_LENGTH];
                                        /* Host to restore to */
165 1   Char              dirName[MAX_STRING_LENGTH];
                                        /* Directory to restore to */
166 1   RestoreInfoPtr    tmpInfo;
                                      /* Pointer to walk list */
167 1   OverwritePolicy   policy;
                                         /* Overwrite policy */
168 1   RestoreTransport  transport;
                                       /* Data Transport type */
169 1   eerrno_ty         eerrno;
                                            /* Error Status */

171 1   /* Do nothing if we're searching, or a restore is in progress */
172 1   if (REST_SearchInProgress() ||
173 1       REST_RestoreInProgress() ||
174 1       (currentWorkItemInfo == NULL))
175 2   {
176 2     return;
177 1   }

179 1   /* Verify that the media is online or the user doesn't care */
180 1   if (!REST_VerifyMedia ())
181 2   {
182 2     return;
183 1   }

185 1   /*
186 1    * Find the client object for this work-item
187 1    * Default to in place host and directory
188 1    */

190 1   /* Use the current info */
191 1   tmpInfo = currentWorkItemInfo;

193 1   /* Find the client for this restoral */
194 1   while ((tmpInfo != NULL) && (tmpInfo->type != REST_Client))
195 2   {
196 2     tmpInfo = tmpInfo->parent;
197 1   }

199 1   /* If we found the client get its name */
200 1   if (tmpInfo != NULL)
201 2   {
202 2     STR_Cpy (hostName, tmpInfo->name);
203 1   }
```

```
205 1    /* Else we have no host ?? */
206 1    else
207 2    {
208 2      STR_Cpy (hostName, "");
209 1    }

211 1    /* Default the directory name to the root directory */
212 1    STR_Cpy (dirName, "/");

214 1    if (REST_GetDestinationInfo ((WinPtr)REST_RestoreWin,
215 1                                 &inPlace,
216 1                                 destHostName,
217 1                                 dirName,
218 1                                 &policy,
219 1                                 &transport))
220 2    {
221 2      /* Submit the restoral */
222 2      if ((eerrno = EDMRST_Submit (GREST_Handle,
223 2                                   destHostName,
224 2                                   policy,
225 2                                   inPlace,
226 2                                   dirName,
227 2                                   transport,
228 2                                   0,
229 2                                    NULL)) != 0)
230 3      {
231 3        REST_DisplayErrnoMessage ((WinPtr)REST_RestoreWin,
232 3                                  NULL,
233 3                                  REST_GetErrorString (
                                                    REST_START_ERROR),
234 3                                  eerrno);
235 2      }
236 2      else
237 3      {
238 3        REST_StartProgress ();
239 2      }
240 1    }
241    }
```